



**IN THE UNITED STATES PATENT AND TRADEMARK OFFICE**

Inventor: Isaac LEVANON *et al.*  
Assignee: 3VDU, Inc.  
Serial No.: 10/035,981  
Filed: December 24, 2001  
Title: System And Methods of Network Image Delivery...  
Examiner: Philip B. Tran  
Group: 2155  
Confirmation No.: 3619  
Attorney Docket: 927/2

This is submitted in response to the Office Action of March 17 2006.

---

**Declaration of Inventors**

Applicants Isaac Levanon and Yoni Lavi being duly sworn, depose and affirm that:

1. We hereby declare that we believe we are the original, first and co-inventors of the subject matter which is claimed herein for which a utility patent is sought on the invention described and claimed in the above-identified application; that we have reviewed and understand the contents of the application, including the claims; and, that we acknowledge our duty to disclose to the PTO information of which we are aware which is material to patentability of this invention as defined in 37 C.F.R. 1.56.
2. The invention as claimed in independent claims 1, 7, 12 and 16 was first defined on paper in October 1999 and reduced to practice by a working computer program in December 1999.

It is established that a working product was in applicant's position by 24 January 2000 in view of the Website presentation evidenced by the screens shots of Exhibit A as filed in Affidavit of 10 January 2006.

Independent claims 1, 7, 12 and 16 are reproduced hereinbelow with part annotations a, b, c, d, e...added, for ease of referencing to the various components:

*Claim 1 relates to a client system for dynamic visualization of image data provided through a network communications channel, said client system comprising:*

*(a) a parcel request subsystem, including a parcel request queue, operative to request discrete image data parcels in a priority order and to store received image data parcels in a parcel data store, said parcel request subsystem being responsive to an image parcel request of assigned priority to place said image parcel request in said parcel request queue ordered in correspondence with said assigned priority; and*

*(b) an parcel rendering subsystem coupled to said parcel data store to selectively retrieve and render received image data parcels to a display memory, said parcel rendering system providing said parcel request subsystem with said image parcel request of said assigned priority.*

Applicant affirms that the parcel request subsystem and the parcel rendering subsystem were reduced to practice by 24 January 2000.

A copy of archived source code, substantially identical to that of the version of January 2000 is appended herewith. This source code shows how the invention was reduced to practice by being programmed into a working system implementation and is proof of Reduction To Practice in that it demonstrates that the claimed invention was programmed and worked. Applicants affirm that the code is substantially that of the version of December 1999.

Specifically, the “Parcel Request Subsystem” (a) consists of multiple consumers, and a single producer. The function “DLDownloaderThread” is executed from four threads running in parallel to the main program (See “DLStartDownloaderThreads” and “DLStopDownloaderThreads”). Each of those threads acts as a consumer of the “Parcel Request Queue”. The priority queue is implemented over a stack of sorted items named “DTL”. The consumers extract items from the top of the stack. The main program acts as

a producer of parcel requests. The function to consume requests from the queue is named DLGetNext.

The function "RenderQuads", in addition to rendering the visible images parcels on the display, also creates requests for parcels that are needed, and evaluates the priority of each request as described in the patent, using function DLAdd. The function DLManager places these requests inside an array, sorts it by priority and replaces the previous stack (Parcel request queue) with the contents of the sorted array.

The priority computation in function DLAdd is performed according to the priority evaluation method explained in the patent application, see paragraphs 59-69 thereof.

The function GenerateTileOffset computes the offset of each requested parcel inside the "Parcel data store" described by the patent. There is an explicit assumption of 2KB per compressed parcel in the data store, possible through use of a fixed-rate compression, which makes the implementation very simple.

The producer side of the Parcel Request Subsystem (explained earlier) in effect shows implementation of *parcel rendering subsystem* (b).

This is also supported by the following routines and functions: DLAdd, DLReset, GenerateOffsetTable, GenerateTileOffset, PolygonSurface, Distort, HttpRequest, TileManagerRender, RecalcTilePlane, DrawTileMap, TexelDensity, TileGetDrawlevel, Fast\_WithinFrustrum, QuadMake, RenderQuads, FrustrumClipper,

All functions in modules Raster.cpp and Tex.cpp

*Claim 7 relates to a portable display client system supporting dynamic visualization of image data provided through a wireless network communications channel, said client system comprising:*

*(c) a display of defined resolution suitable for visual presentation of a graphical image, said display including video memory for storing image data representative of said graphical image;*

*(d) a network interface coupleable to a wireless network through which*

*to request and receive image data parcels;*

*(e) an image parcel data store providing for the storage of image data parcels;*

*(f) navigation controls providing input information defining a point of view location relative to said graphical image; and*

*(g) a processor coupled to said video memory, network interface, image parcel data store, and navigational controls, said processor operative to selectively request image data parcels of determined resolution through said network interface in a priority order computed relative to the defined resolution of said display.*

Applicant asserts that the invention of claim 7 was reduced to practice in December 1999. All components related to in claim 7 were implemented in the software prior to January 2000. Fragments of source code given herewithin show this implementation.

The display of defined resolution suitable for visual presentation of a graphical image (c) is supported by the FlipWindow function.

The Network interface (d) is utilized by the function Http11Request, which is used to retrieve image parcels through the HTTP/1.1 protocol.

The function “Dynamic\_Camera” implements various navigation commands which respond to various keystrokes. This supports (f) *the navigation controls providing input information defining a point of view location relative to said graphical image.*

The hardware specified in (c), (e), (g) by virtue of being hardware, cannot be directly shown in any source code of course. Nevertheless, evidence that these components were available in January 2000 is given through inference and via the OS API and runtime library calls. As evidenced by calls to functions such as GetAsyncKeyState, BitBlt and socket, the program interacted with these hardware elements, available in January 2000 and described in the provisional patent specification filed in December of that year.

*Claim 12 relates to a client system providing for the three-dimensional viewing of an image relative to a dynamic viewpoint, said client system comprising:*

*(h) display memory corresponding to a two-dimensional display of defined resolution;*

*(i) a network interface through which image data parcels of respective image parcel resolution can be requested and received;*

*(j) navigational controls providing for the definition of an image viewpoint; and*

*(k) a processor operative to progressively render image data parcels into said display memory and to progressively request image data parcels through said network interface in a priority order determined with respect to the defined resolution of said two dimensional display and the location of said image viewpoint.*

Applicant asserts that the invention of claim 12 was similarly reduced to practice by December 1999.

(h) is evidenced by InitDevice and FlipWindow

Further evidence that these components were available in January 2000 is given through inference and via the OS API and runtime library calls. As evidenced by calls to functions such as GetAsyncKeyState, BitBlt and socket, the program interacted with these hardware elements, available in January 2000 and described in the provisional patent specification filed in December of that year.

The Network interface (i) is utilized by the function Http11Request, which is used to retrieve image parcels through the HTTP/1.1 protocol.

The *navigation controls providing input information defining a point of view location relative to said graphical image* (j) are supported by the function "Dynamic\_Camera" which implements various navigation commands in response to various keystrokes.

The hardware specified in (k), by virtue of being hardware, cannot be directly shown in any source code.

We affirm that all components related to in claim 12 were implemented in the software by 24 June 2000. The fragments of source code given hereabove support this affirmation.

*Claim 16 relates to a method of supporting dynamic visualization of image data transferred through a communications channel, said method comprising the steps of:*

*(l) determining, in response to user navigational commands, a*

*viewpoint orientation with respect to an image displayed within a three-dimensional space;*

*(m) requesting, in a priority order, image parcels renderable as*

*corresponding regions of said image, each said image parcel having an*

*associated resolution, wherein said priority order is determined to provide a*

*progressive regional resolution enhancement of said image as each said image parcel is rendered;*

*(n) receiving a plurality of image parcels through said communications channel; and*

*(o) rendering said plurality of image parcels to provide said image.*

(l) is supported by the function "Dynamic\_Camera" which implements various navigation commands which respond to various keystrokes,

(m) is implemented in the producer side of the Parcel Request Subsystem, by functions including:

DLAdd, DLReset, GenerateOffsetTable, GenerateTileOffset, , PolygonSurface, Distort, Http11Request

(n) is implemented in the function `Http11Request`, which is used to retrieve image parcels through the HTTP/1.1 protocol

See also `DLReceive` which processes the output from the Parcel Request Subsystem, creating a 64 by 64 texture map by applying the image decoder on the received data block, using the function `DecodeFxt1`. The FXT1 texture compression format has a fixed rate compression of 4 bits per pixel, yielding 2KB of data per each compressed 64 by 64 image parcel. This matches the compression rate for parcels specified in the patent application.

(o) is implemented by the functions “`TileManagerRender`” and its subroutines (“`DrawTileMap`”, “`QuadMake`” “`RenderQuads`”, “`Fast_WithinFrustrum`”, “`SpliceVertex`”, “`FrustrumClipper`” among others) which render the image data to the system display. See also `TileRecalcPlane`, `TexelDensity`, `TileGetDrawlevel`, All functions in modules `Raster.cpp` and `Tex.cpp`

Applicant asserts that all components related to in claims 1, 7, 12, 16 were implemented in the software in December 1999, prior to publishing of the Website Presentation on the Internet as shown by Exhibit B.

As evidenced by Exhibit C, GA Central Executive Summary (appended herewithin as Exhibit Ci) was written up by 20 March 2000 and Power Point Presentation GA Central2.ppt (Exhibit Cii) was written up by 13 May 2000. These documents provide further proof that the working system, including code appended hereto, were in applicant’s position prior to the filing date of US 6,671,424 to Skoll, which was filed in July 2000.

Exhibit D is a screen shot extracted from Exhibit Cii, which, as evidenced by Exhibit C, was written on or before 13 May 2000. Applicant affirms that the screen shot shown in Exhibit C was obtained by running the program that embodies the invention as claimed in claims 1, 7, 12 and 16, back in December 1999. This supports Applicant’s claim that the invention as shown in source code appended herewith was in their possession in December 1999 which is six months prior to the filing date of US 6,671,424 to Skoll, which was filed in July 2000.

Exhibit E and G are screen captures from December 1999 that shows that the program whose source code is appended herewith was indeed up and running at that time.

It will be noted that although the code appended may have been minorly revised during maintenance, and may not perfectly match the initial implementation however it shows that the present invention as claimed was indeed reduced to practice.

Applicant has thus demonstrated that the invention as claimed in claims 1, 7, 12 and 16 was reduced to practice prior to July 2000 and respectfully submits that US 6,671,424 Skoll, which was filed in July 2000, is not prior art to these claims.

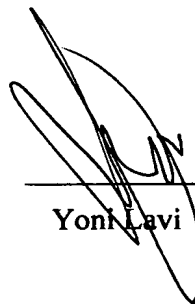
If claims 1, 7, 12 and 16 are inventive, then dependent claims 2-6, 8 to 11 and 17 to 20 are inventive as well. We respectfully submit that claims 1 to 11 and 16 to 20 are thus allowable, and request that Examiner withdraws his objections of March 17 2006 and allows the claim set of USSN 10/035,981 as filed.

3. The evidence submitted is sufficient to establish a reduction to practice of the invention in the US or a NAFTA or WTO member country prior to the effective date of the Robotham reference.

4. We hereby declare that all statements made herein are of our own knowledge and are true. All statements made on information and belief are believed to be true. All statements are made with knowledge that willful false statements and the like so made are punishable by fine or imprisonment or both, under Section 1001 of Title 18 of the United States Code and that such willful statements may jeopardize the validity of the application or any patent issued thereon.



Isaac Levanon



Yoni Lavi

Dated: June 13, 2006



## EXHIBITS

### EXHIBIT A

Microsoft Internet Explorer

Check out | Shopping cart | Great values | Customer service

Search  Go

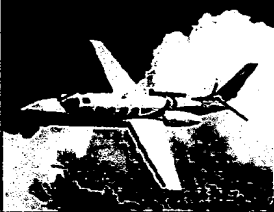
Marketplace | Information | Community | Air Network

Product List | Search Results | Great Values

Product Name | Price | Go to | Add to cart

1999 AeroStar Jet	\$115,000.00	Go to	Add to cart
2000 AeroStar Jet	\$135,000.00	Go to	Add to cart
2001 AeroStar Jet	\$155,000.00	Go to	Add to cart
2002 AeroStar Jet	\$175,000.00	Go to	Add to cart

Product | Detail Information | Customer Service



The AeroStar Jet will cruise at 400 knots, climb to 41,000 feet with a cabin altitude of 10,000 feet, and have a range exceeding 1,200 nautical miles.

City:  ☐

Cond.: ☒ New ☐

**AEROSTAR**

**Pilots** Save 30%-70%

**The First of Its Kind** Portable!

**Find your Music HERE** CD/DVD

**Health Watch** FREE

**LOWEST PRICES** FREE

**WIN \$3,333** Shopping Spree!

**ORDER THE MATRIX** amazon.com

[illegible]

# Exhibit C

WinZip (Evaluation Version) - Patent_Info.zip						
File Actions Options Help						
Name	Type	Modified	Size	Ratio	Packed	Path
GA Central - Executive Summary.doc	Microsoft Word Document	3/20/2000 6:02 PM	1,689,600	25%	1,260,...	
GA Central2.ppt	Microsoft PowerPoint Presentation	5/13/2000 12:20 AM	836,096	42%	486,612	

Name	Type	Modified	Size
GA Central - Executive Summary.doc	Microsoft Word Document	3/20/2000 6:02 PM	1,689,600
GA Central2.ppt	Microsoft PowerPoint Presentation	5/13/2000 12:20 AM	836,096

**GA Central**  
**.COM**

# **The FlyOver™**

## **A 3D Visualization Technology**

**Proprietary 3D  
engine streaming  
high-resolution  
image (satellite or  
aerial) over the  
Internet at about  
4k/sec. with six  
degrees of  
freedom**

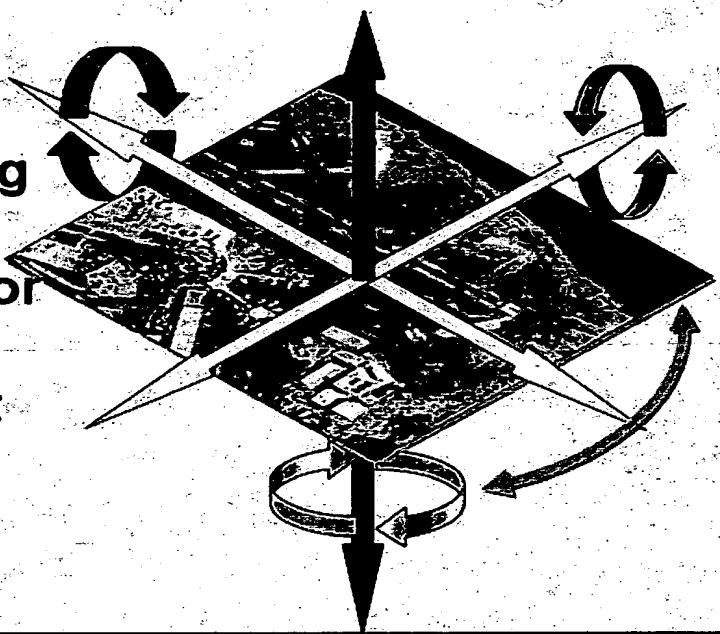
A diagram illustrating the 3D visualization technology. It features a central satellite image of a city, tilted at an angle. Six arrows originate from the center of the image, pointing in different directions to represent the six degrees of freedom: up, down, left, right, forward, and backward. Each arrow is accompanied by a circular arrow indicating rotation around that axis.

Exhibit E

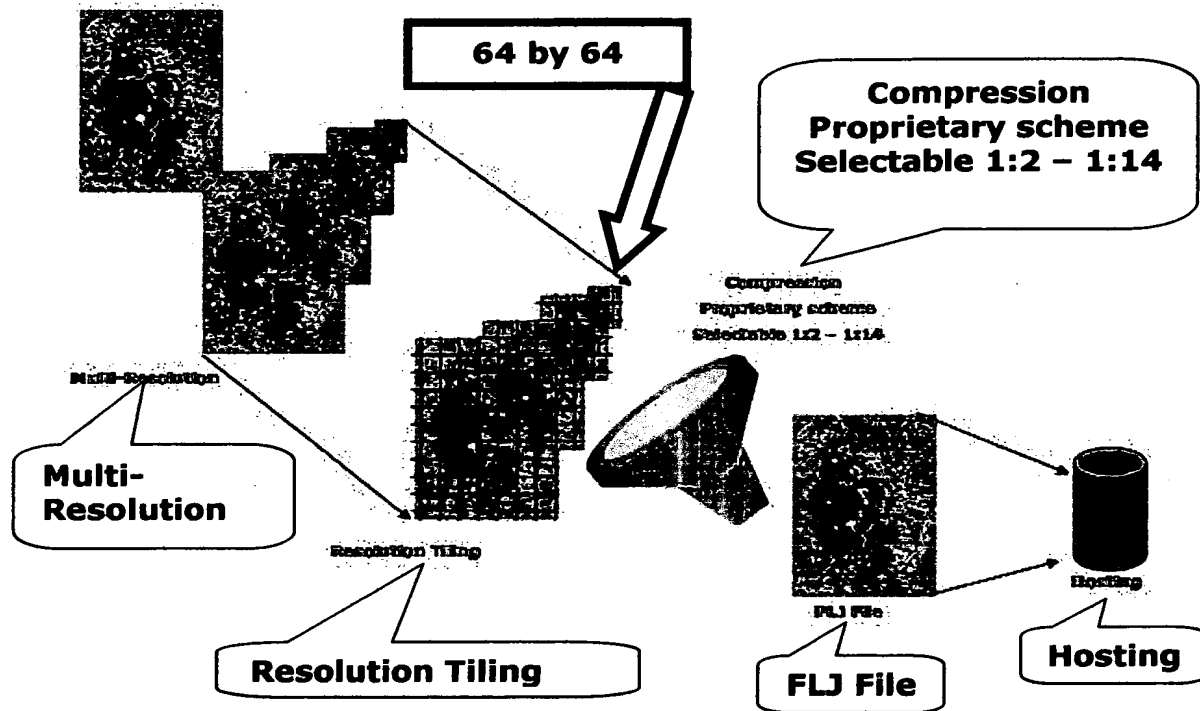
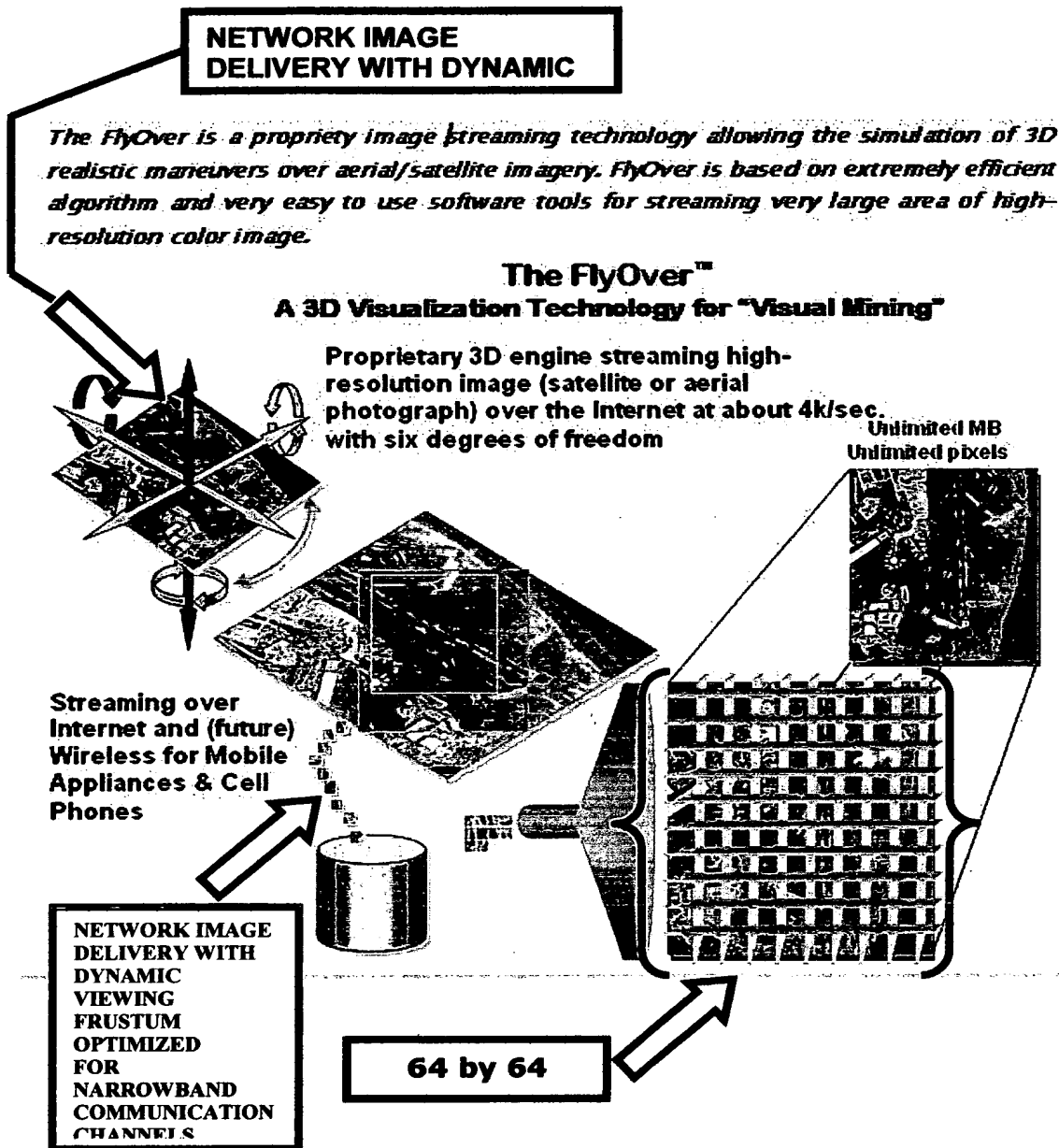


Exhibit F



**64 by 64**

**Incremental Streaming**

**Decompress**

**Rebuild tiles and resolutions**

**Request multi-tiles in multi-Resolutions**

**Smart Agent**  
tightly coupled

**3D Engine**

**Request for image parcel**

**Real-time filters**

**Extremely thin software based real-time smart streaming agent tightly coupled to efficient 3D engine, decoder and real-time filters  
On Any Digital Device**

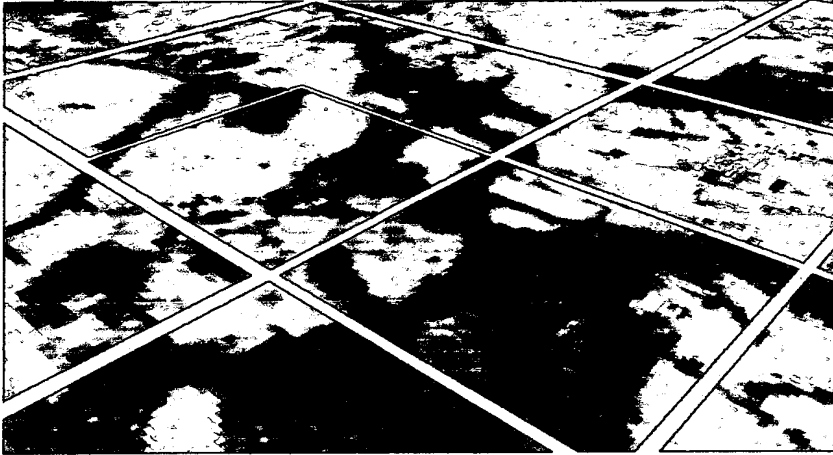
**3D Engine**

**Smart**

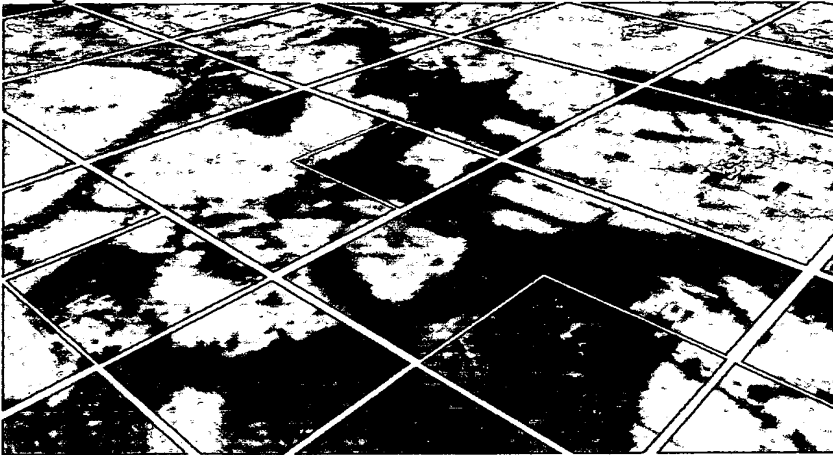
**Request multi-tiles of multi-Resolutions**

**Exhibit H**

**Image 1**



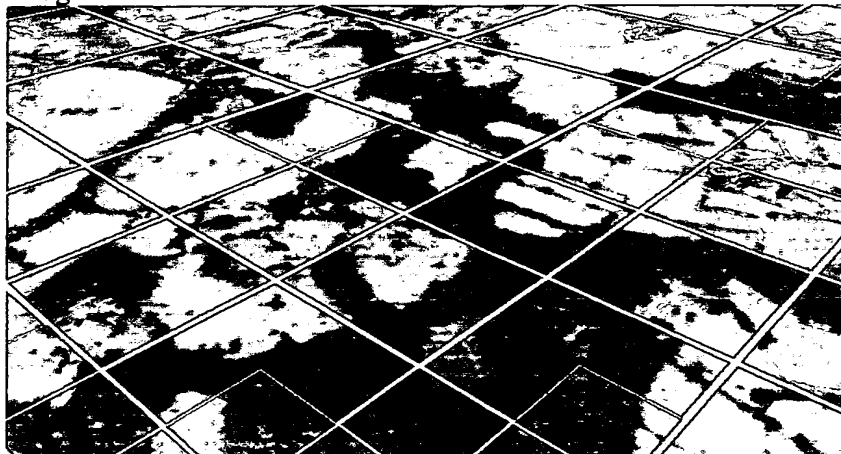
**Image2**



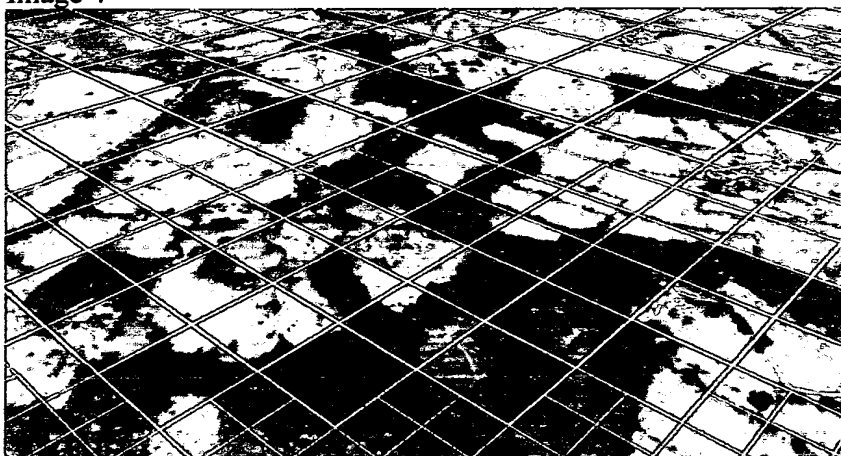


**Exhibit H continued**

**Image 3**



**Image 4**



## APPENDIX (I)

### Source code listing

Module: MapTiles.cpp

---

```
// Download recieve: this is called by the Downloader
threads when they finish downloading
// one of the map tiles. It decodes the FXT1 image and
generates a texture which is
// inserted to the texture cache.
void DLRecieve(MapTile *MT, byte *pBuf)
{
    Image *Img = new Image;

    DecodeFxt1(pBuf, Img);

    if (!Img->pData)
    {
        printf("*** ARPA ALARMS: Failed to D/L texture
from map server\n");
        return;
    }

    if ((Img->xRes > g_TM.TileSize) ||
        (Img->yRes > g_TM.TileSize))
    {
        printf("*** ARPA ALARMS: Downloaded image is too
large (incorrect TM setup?)\n");
        //return;
    }

    if ((Img->xRes < g_TM.TileSize) ||
        (Img->yRes < g_TM.TileSize))
    {
        ImagePadding(Img);
    }

#ifdef _FLY_OVER_WIN_CE
    __MAPPER__Tile4(Img); // ??? if i want to use the ref-
resterizer put this line remark. and hit R
#endif

    // before removing eyal+gilad memory trick

    /*    word *w = new word[64*64*5];
        //memcpy(w, Img-
>pData, sizeof(word)*64*64);
```

```

        memcpy(w+64*64*4, Img-
>pData, sizeof(word)*64*64);
        DWORD I;
        for(I=0; I<64; I++)
            memcpy(w+256*I, Img-
>pData+64*I, sizeof(word)*64);

        delete Img->pData;

        Img->pData=w;
        // Texture -> 256x64
        //__MAPPER_ImagePadding(Img);
        //__MAPPER_Align64(Img);

        Img->pData += 64*64*4;
        __MAPPER_Tile4(Img);
        Img->pData -= 64*64*4;*/

    // end - before removing eyal+gilad memory trick

    // tiling shit
    TTexStorage_Add(MT, Img->pData);
    delete Img;
    //Insert2TCache(MT, Img->pData);
    //delete Img->pData;
    //MT->Texture = Img; //SYNCED! so engine wont try to
use a non-padded texture
/*    if (MT->Flags & MT_JIT)
    {
        int wtf = 1;
    }*/

    //MT->Flags &= ~MT_Downloading;
}

// Resets the download request list
static void DLReset()
{
    dword i;
    for(i=0; i<g_TM.DLNumRequests; ++i)
    {
        g_TM.DLRequests[i].MT->DR = NULL;
    }
    g_TM.DLNumRequests = 0;
}

```

```

// Garage Operation. (Calculates offsets inside map tile
master index file)
static void GenerateOffsetTable()
{
    // BASE OFFSET: 1K
    const dword BASE_OFFS = 1024;
    // IMAGE CELL SIZE (TSxTSx16bit using FXT1): 4bit *
TS^2
    const dword BASE_HEADER = 16;
    g_TM.IMAGE_CELL_SIZE =
((g_TM.TileSize*g_TM.TileSize)>>1)+BASE_HEADER;

    dword I,OFFS = BASE_OFFS;

    for(I=0;I<g_TM.MaxLevel;I++)
    {
        // LEVEL OFFSET: #Tiles(T|Lvl(T)=L) *
IMAGECELLSIZE
        dword S = g_TM.TSize[I];
        dword X = (g_TM.XSize+S-1)/S; //divide + round up
        dword Y = (g_TM.YSize+S-1)/S; //divide + round up
        g_TM.XTiles[I] = X;
        g_TM.Offs[I] = OFFS;

        OFFS += X*Y*g_TM.IMAGE_CELL_SIZE;
    }
}

// calculates offset inside map tile master index file,
given the Texture's level and
// (X,Y) in that level's tile grid.
static dword GenerateTileOffset(dword Level,dword X,dword
Y)
{
    return g_TM.Offs[Level] + (X + Y *
g_TM.XTiles[Level])*g_TM.IMAGE_CELL_SIZE;
}

// Distorts coordinates to create the fisheye effect needed
for prioritizing
// cells at the view's center (focus) over peripheral
regions
// dfactor: distortion factor [0,1]. dfactor=1 has no
distortion effect
static inline float Distort(float x,float dfactor = 0.35)
{
    if (x>0)

```

```

    {
        return (float)pow(x,dfactor);
    } else {
        return -(float)pow(-x,dfactor);
    }
    return 0.0;
}

// Computes the surface of the polygon currently stored in
the frustrum clipper
// can work either pre or post clipping
static float PolygonSurface()
{
    long i;
    float s = 0.0f;
    Vertex *A,*C;
    A = C_Prim[0];

    float ax = Distort(A->PX / (float)(g_xRes*0.5f) -
1.0f),
        ay = Distort(A->PY / (float)(g_yRes*0.5f) -
1.0f);
    C = C_Prim[1];
    float cx = Distort(C->PX / (float)(g_xRes*0.5f) -
1.0f),
        cy = Distort(C->PY / (float)(g_yRes*0.5f) -
1.0f);
    float bx,by;
    for(i=2;i<C_NumVtx;++i)
    {
        bx = cx;
        by = cy;
        C = C_Prim[i];
        cx = Distort(C->PX / (float)(g_xRes*0.5f) -
1.0f);
        cy = Distort(C->PY / (float)(g_yRes*0.5f) -
1.0f);
        s += (ax * by - bx * ay) +
            (cx * ay - ax * cy) +
            (bx * cy - cx * by);
    }
    return (float)fabs(s);
}

// Add maptile to the list of requested downloads.
// DLev (Draw level) is used to determine the priority of
the request.

```

```

// Draw level should be the texture detail level in which
the map tile would have
// been drawn if all the textures were available
(mipmapping limit)
static void DLAdd(Quad *Q, MapTile *MT)
{
    long DLev = Q->drawlev;
    /* if (!MT)
    {
        int gagaga=1;
    }*/
    if (MT->Flags&MT_Downloading) return; //I know all
about the Zerg, Gerard.

    dword I;
    if (MT->DR) {
        I = MT->DR - g_TM.DLRequests;
    } else {
        if (g_TM.DLNumRequests==g_TM.DLMaxRequests)
        {
            g_TM.DLMaxRequests+=DLMaxInitialRequests;
            g_TM.DLRequests = (DownloadRequest
*)realloc(g_TM.DLRequests,g_TM.DLMaxRequests*sizeof(Downloa
dRequest));
        }
        I = g_TM.DLNumRequests++;
        g_TM.DLRequests[I].MT = MT;
        g_TM.DLRequests[I].FOffset =
GenerateTileOffset(MT->Level,MT->TX,MT->TY);
        g_TM.DLRequests[I].Priority = 0;
        MT->DR = g_TM.DLRequests+I;
    }

    //GenerateTileFilename(g_TM.DLRequests[I].Name,MT-
>Level,MT->TX,MT->TY);
    /* if (MT->Level==4)
    {
        int debug_me_now = 1;
    }*/

    //g_TM.DLRequests[I].Priority = 128+MT->Level*2-DLev;
    g_TM.DLRequests[I].Priority += PolygonSurface(); // *
(1<<((DLev-MT->Level) << 1));
}

```

```

// Creates child nodes for parameter maptile. This is
// called whenever the tile
// has a draw level greater than its own level. [and its
// own texture is available]
static dword MapTileExpand(MapTile *MT)
{
    if (MT->Flags&MT_Expanded) return 1; //already
expanded
    if (!MT->Texture) return 0; // no texture recieved yet

    MT->Flags|=MT_Expanded;
    dword Level = MT->Level+1;

    if (Level==g_TM.MaxLevel) return 1;

    dword xE = MT->X+g_TM.TSize[Level];
    dword yE = MT->Y+g_TM.TSize[Level];
    dword xOK = xE<g_TM.XSize;
    dword yOK = yE<g_TM.YSize;

    MT->M00 = new MapTile;
    memset(MT->M00,0,sizeof(MapTile));
    MT->M00->Level = Level;
    MT->M00->X = MT->X;
    MT->M00->Y = MT->Y;
    MT->M00->TX = MT->TX<<1;
    MT->M00->TY = MT->TY<<1;
    MT->M00->Parent = MT;

    if (xOK)
    {
        MT->M01 = new MapTile;
        memset(MT->M01,0,sizeof(MapTile));
        MT->M01->Level = Level;
        MT->M01->X = xE;
        MT->M01->Y = MT->Y;
        MT->M01->TX = (MT->TX<<1)+1;
        MT->M01->TY = MT->TY<<1;
        MT->M01->Parent = MT;
    }

    if (yOK)
    {
        MT->M10 = new MapTile;
        memset(MT->M10,0,sizeof(MapTile));
        MT->M10->Level = Level;
    }
}

```

```

        MT->M10->X = MT->X;
        MT->M10->Y = yE;
        MT->M10->TX = MT->TX<<1;
        MT->M10->TY = (MT->TY<<1)+1;
        MT->M10->Parent = MT;
    }

    if (xOK&&yOK)
    {
        MT->M11 = new MapTile;
        memset(MT->M11,0,sizeof(MapTile));
        MT->M11->Level = Level;
        MT->M11->X = xE;
        MT->M11->Y = yE;
        MT->M11->TX = (MT->TX<<1)+1;
        MT->M11->TY = (MT->TY<<1)+1;
        MT->M11->Parent = MT;
    }

    return 1;
}

// should be called for each frame. determines the plane
// equation
// of the map, transformed to camera space.
void TileRecalcPlane()
{
    // fundamental information: in world coordinates,
    // Origin is always at (0,0,0), the normal is always
    (0,1,0)
    // XAxis is (1,0,0) and YAxis is (0,0,1).

    // Note! Image is inverted on the Y Axis, so Origin is
    shifted and axis is flipped

    Vector3 U;
    float *VM = g_TM.Cm->VM;
    // calculate new origin
    VectorScale(&g_TM.Cm->Position,-1.0f,&U);
    MatrixXVector(VM,&U,&g_TM.Origin);

    // calculate new axes
    VectorMake(&g_TM.XAxis,VM[0],VM[3],VM[6]); // VM *
    (1,0,0)
    VectorMake(&g_TM.Norm ,VM[1],VM[4],VM[7]); // VM *
    (0,1,0)

```



```

        VectorMake(&g_TM.YAxis,-VM[2],-VM[5],-VM[8]); // VM *
(0,0,1)

        VectorSelfScale(&g_TM.XAxis,g_TM.TexelSize);
        VectorSelfScale(&g_TM.YAxis,g_TM.TexelSize);
        VectorScale(&g_TM.YAxis,-
(float)g_TM.YSize,&U); //g_TM.TSize[0]
        VectorSelfAdd(&g_TM.Origin,&U);

        g_TM.PO =-VectorDot(&g_TM.Norm,&g_TM.Origin);
}

// Determines the size one map texel at location (x1,y1) on
the map has on the display after projection
float TexelDensity(float x1,float y1)
{
    Vector3 V;
    VectorMake(&V,g_TM.Origin.x + x1 * g_TM.XAxis.x + y1 *
g_TM.YAxis.x,
                g_TM.Origin.y + x1 * g_TM.XAxis.y +
y1 * g_TM.YAxis.y,
                g_TM.Origin.z + x1 * g_TM.XAxis.z +
y1 * g_TM.YAxis.z);

    float z = V.z*g_TM.TileSize/g_TM.TSize[0];
    float z2 = z*z;
    VectorNorm(&V);
    float ca = -VectorDot(&g_TM.Norm,&V);

    return g_TM.Cm->VP->Mipfactor * ca / z2;
}

// efficient Log2 operation on 32bit float (single
precision) type
long Log2(float x)
{
    long dwrep = *(long *)(&x);
    long exponent = (dwrep>>23)-127;

    return exponent;
}

// returns highest mipmapping level on a transformed
Quadritheral,
// Sucks. should be based on pix/tex Area ratio computation
void TileGetDrawlevel(MapTile *MT,dword *h)
{

```

```

        // calculate camera [x,y] on tile plane coordinate
        float ratio = -1.0f/(g_TM.TexelSize*g_TM.TexelSize);
        long cx =
(long) (ratio*VectorDot(&g_TM.Origin,&g_TM.XAxis));
        long cy =
(long) (ratio*VectorDot(&g_TM.Origin,&g_TM.YAxis));

        long x1 = MT->X;
        long y1 = MT->Y;

        long x2 = MT->X+g_TM.TSize[MT->Level];
        long y2 = MT->Y+g_TM.TSize[MT->Level];

        if (cx<x1) cx = x1;
        else if (cx>x2) cx = x2;

        if (cy<y1) cy = y1;
        else if (cy>y2) cy = y2;

        // Nearest (x1,y1) Furthest (x2,y2)
        long lvl = Log2(TexelDensity((float)cx,(float)cy))>>1;
// log4(dens)
        long sl = lvl>=(long)g_TM.MaxLevel ? g_TM.MaxLevel-1 :
lvl;
        *h = sl>0 ? sl : 0;
    }

// This is a recursive function which builds up the
Rendering information.
// Either it recurses or calls QuadMake, thus rendering
the entire visible
// portion of the map.
void DrawTileMap(MapTile *MT,Vector3 *A,Vector3 *B,Vector3
*C,Vector3 *D)
{
    dword high;
    TileGetDrawlevel(MT,&high);
    // LA-Z-Eval: if high>MT->Level, maptileexpand will be
called,
    // and if failed, entire quad will be drawn without
expanding
    if (high<=MT->Level||(!MapTileExpand(MT)))
    {
        //if (high>MT->Level) high=MT->Level;
        // draw Tile as a Quad
        float xFrac = 1.0f,yFrac = 1.0f;
        if (MT->X+g_TM.TSize[MT->Level]>g_TM.XSize)

```

```

        xFrac = (float)(g_TM.XSize-MT-
>X)/(float)g_TM.TSize[MT->Level];
        if (MT->Y+g_TM.TSize[MT->Level]>g_TM.YSize)
            yFrac = (float)(g_TM.YSize-MT-
>Y)/(float)g_TM.TSize[MT->Level];

        QuadMake(high,MT,A,B,C,D,xFrac,yFrac);
        return;
    }

    Vector3 AB,BC,CD,DA,ABCD;
    VectorAvg(A,B,&AB);
    VectorAvg(B,C,&BC);
    VectorAvg(C,D,&CD);
    VectorAvg(D,A,&DA);
    VectorAvg(&AB,&CD,&ABCD);
    if (MT->M00) DrawTileMap(MT->M00,A,&AB,&ABCD,&DA);
    if (MT->M01) DrawTileMap(MT->M01,&AB,B,&BC,&ABCD);
    if (MT->M11) DrawTileMap(MT->M11,&ABCD,&BC,C,&CD);
    if (MT->M10) DrawTileMap(MT->M10,&DA,&ABCD,&CD,D);
}

// stores the ngon visible from frustrum on map
struct WF_Visible_NGon
{
    Vector2 VA[12];
    dword NumVtx;
};

WF_Visible_NGon FWF_;

// initializes the fast within-frustrum test routine
void Fast_WithinFrustrum_Init(Viewport *VP,Vector3 **V)
{
    // intersect entire region with viewport
    // by calling f-clipper on a generated poly
    // somehow remove the call to drawer
    // XFORM V
    Vertex VA[4];
    long i;
    for(i=0;i<4;i++)
    {
        VA[i].TPos.x = g_VP.CX * V[i]->z + g_VP.K * V[i]-
>x;
        VA[i].TPos.y = g_VP.CY * V[i]->z - g_VP.K * V[i]-
>y;
        VA[i].TPos.z = V[i]->z;
    }
}

```

```

        VA[i].RZ = 1.0f/VA[i].TPos.z;
        VA[i].PX = VA[i].TPos.x * VA[i].RZ;
        VA[i].PY = VA[i].TPos.y * VA[i].RZ;
        VertexFlags(VA+i);
        C_Buf[i] = VA+i;
    }

    C_NumVtx = 4;

    C_Flags =
Vtx_Visibility&(VA[0].Flags|VA[1].Flags|VA[2].Flags|VA[3].F
lags);

    FrustrumClipper(NULL);

    // copy to FWF_
    FWF_.NumVtx = C_NumVtx;
    float rK = 1.0f/g_VP.K;
    for(dword I=0;I<C_NumVtx;I++)
    {
        // memcpy(FWF_.VA+I,C_Prim[I],sizeof(Vertex));
        // Inverse-Transform perspective and 3d->map
coordinate
        Vector3 V;
        V.z = 1.0f/C_Prim[I]->RZ;
        float z = rK*V.z;
        V.x = (C_Prim[I]->PX-g_VP.CX)*z;
        V.y = (g_VP.CY-C_Prim[I]->PY)*z;
        //now V is a position in 3d[camera-space] (i
hope)

        VectorSelfSub(&V,&g_TM.Origin);
        FWF_.VA[I].x =
VectorDot(&V,&g_TM.XAxis)*g_TM.rTexelSize;
        FWF_.VA[I].y =
VectorDot(&V,&g_TM.YAxis)*g_TM.rTexelSize;
        //IFontWrite(IFNT_STD, 10, 50+20*I, "Delimiter
NGON (%f,%f)",FWF_.VA[I].x,FWF_.VA[I].y);
    }
    // complete cycle
    FWF_.VA[C_NumVtx].x = FWF_.VA[0].x;
    FWF_.VA[C_NumVtx].y = FWF_.VA[0].y;
}

dword Within_NGon(Vector2 *U)
{
    dword I,J=0;

```

```

Vector2 n,d;
for(I=0;I<FWF_.NumVtx;I++)
{
    n.x = FWF_.VA[I ].y - FWF_.VA[I+1].y;
    n.y = FWF_.VA[I+1].x - FWF_.VA[I ].x;
    d.x = U->x-FWF_.VA[I].x;
    d.y = U->y-FWF_.VA[I].y;
    if (n.x*d.x+n.y*d.y<0.0f) J+=1<<I;
}
return J;
}

dword Fast_WithinFrustrum(MapTile *MT)
{
    // square:
    //MT->X,MT->Y : MT->X+g_TM.TSize[MT->Level],...
    // NGON is supposed to be CCW,so, at least one
    // of the points is supposed to be inside ngon
    // WRONG! this _has_ to clip the square vs the ngon
    // (arg) or reject squares that are all on the
negative
    // side of one of the edges of the ngon
    dword l = MT->Level;
    Vector2 U,V;
    V.x = (float)MT->X*g_TM.TexelSize;
    V.y = (float)MT->Y*g_TM.TexelSize;
    float edgelen = g_TM.TSize[l] * g_TM.TexelSize;
    dword f1,f2,f3,f4; // clipping flags
    if (!(f1=Within_NGon(&V))) return 1;
    U.x = V.x+edgelen;
    U.y = V.y;
    if (!(f2=Within_NGon(&U))) return 1;
    U.y += edgelen;
    if (!(f3=Within_NGon(&U))) return 1;
    U.x = V.x;
    if (!(f4=Within_NGon(&U))) return 1;
    return (!(f1&f2&f3&f4));
}

// Generates a Quadritheral for rendering. xFrac and yFrac
are used to build partial tiles.
inline void QuadMake(dword drawlevel,MapTile *MT,Vector3
*A,Vector3 *B,Vector3 *C,Vector3 *D,float xFrac,float
yFrac)
{
    // Verify the Quadritheral is within the viewing
frustrum.

```

```

    if (!Fast_WithinFrustrum(MT)) return;

    Quad *Q = QuadGet();
    Q->MT = MT;

    long texlevel = (drawlevel>MT->Level) ? MT->Level :
drawlevel;

    long leveldiff = MT->Level - texlevel;
    long levelpenalty;
    long i,j;

    MapTile *PMT = MT; // Parent Maptile

    for(i=0;i<leveldiff;++i) PMT = PMT->Parent;

    // Quad should use PMT's texture.
    // if it does not exist, we keep going up the tree
until we find a maptile with
    // a texture (root should always have a texture)
    while ((!PMT->Texture)|| (PMT->Flags & MT_Downloading))
    {
        if (PMT->Parent)
        {
            PMT=PMT->Parent;
            ++leveldiff;
            --texlevel;
        } else {
            // Fatal Error: no texture available
            --NumQuads;
            return;
        }
    }

    // Calculate mapping coordinates
    float rdtex = 1.0f/g_TM.TSize[texlevel];

    float tl = 1.0f/((float)(1<<leveldiff));
    float u = (MT->X - PMT->X)*rdtex;
    float v = (MT->Y - PMT->Y)*rdtex;

    Vector3 Pos[4]; // Position vectors, scaled using
xFrac and yFrac
    VectorCopy(Pos ,A);
    VectorCopy(Pos+1,B);
    VectorCopy(Pos+2,C);
    VectorCopy(Pos+3,D);

```

```

VectorLERP(Pos+1,Pos ,1.0f-xFrac);
VectorLERP(Pos+2,Pos+3,1.0f-xFrac);
VectorLERP(Pos+3,Pos ,1.0f-yFrac);
VectorLERP(Pos+2,Pos+1,1.0f-yFrac);

Vertex *VA[4] = {&Q->A,&Q->B,&Q->C,&Q->D};
Vertex *Vtx;
Vector3 *V = Pos;
// set up mapping coordinates
float utl_ = xFrac * t1 * (LEFTU/64.0f);
float vtl_ = yFrac * t1 * (LEFTU/64.0f);
float utl = xFrac * t1 * (RIGHTU/64.0f);
float vtl = yFrac * t1 * (RIGHTU/64.0f);
#ifdef HYPERBOLIC_MAPPING
VA[0]->U = u+utl_+PMT->TX;
VA[0]->V = v+vtl_+PMT->TY;
VA[1]->U = u+utl_+PMT->TX;
VA[1]->V = v+vtl_+PMT->TY;
VA[2]->U = u+utl_+PMT->TX;
VA[2]->V = v+vtl_+PMT->TY;
VA[3]->U = u+utl_+PMT->TX;
VA[3]->V = v+vtl_+PMT->TY;
#else
VA[0]->U = u+utl_;
VA[0]->V = v+vtl_;
VA[1]->U = u+utl_;
VA[1]->V = v+vtl_;
VA[2]->U = u+utl_;
VA[2]->V = v+vtl_;
VA[3]->U = u+utl_;
VA[3]->V = v+vtl_;
#endif
// Transform Vertices and calculate clipping flags
for(i=0;i<4;i++,V++)
{
    Vtx = VA[i];
    Vtx->TPos.x = g_VP.CX * V->z + g_VP.K * V->x;
    Vtx->TPos.y = g_VP.CY * V->z - g_VP.K * V->y;
    Vtx->TPos.z = V->z;
    Vtx->RZ = 1.0f/Vtx->TPos.z;
    Vtx->PX = Vtx->TPos.x * Vtx->RZ;
    Vtx->PY = Vtx->TPos.y * Vtx->RZ;
    Vtx->UZ = Vtx->U*Vtx->RZ;
    Vtx->VZ = Vtx->V*Vtx->RZ;
    VertexFlags(Vtx);
}

```

```

    // Set up texturing parameters
    Q->Tex = PMT->Texture->Tex;
    Q->drawlev = drawlevel;
    Q->availlev = PMT->Level;//Q->MT->Level;//Parent-
>Level;

    // Update data integrity
    g_TM.DataIntegrityNorm += pow2fast(-2*(MT->Level + Q-
>drawlev));
    g_TM.DataIntegrity += pow2fast(-2*(MT->Level + Q-
>drawlev-Q->availlev ));
    g_TM.DataIntegrityMax += pow2fast(-2*MT->Level);

    // Rasterize Q on Expansion map
    long qlvl = Q->MT->Level;
    long qscanline = (1 << (g_TM.MaxLevel-1)) + 2;
    long qlength = 1 << ((g_TM.MaxLevel-1) - qlvl);
    long qposx = qlength * Q->MT->TX + 1;
    long qposy = qlength * Q->MT->TY + 1;
    byte * qptra = g_TM.ExpansionMap + qposx + qposy *
qscanline;
    for(j=0;j<qlength;j++,qptra += qscanline)
        memset(qptra, qlvl, qlength);
}

// Renders all of the generated Quadritherals.
void SpliceVertex(Vertex *A,Vertex *B,float t,Vertex *V)
{
    float s=1.0f-t;
    V->TPos.x = A->TPos.x * s + B->TPos.x * t;
    V->TPos.y = A->TPos.y * s + B->TPos.y * t;
    V->TPos.z = A->TPos.z * s + B->TPos.z * t;

    V->U = A->U * s + B->U * t;
    V->V = A->V * s + B->V * t;

    V->RZ = 1.0f/V->TPos.z;
    V->PX = V->TPos.x * V->RZ;
    V->PY = V->TPos.y * V->RZ;

    V->UZ = V->U * V->RZ;
    V->VZ = V->V * V->RZ;

```



```

        // kickass
        if (A->Flags | B->Flags)
            VertexFlags(V);
        else
            V->Flags = 0;
    }
void RenderQuads()
{
    Quad *Q,*QE;
    MapTile *MT;

    if (!NumQuads) return;

    long i,j;

    dufactor= Quads->availlev;

    static Vertex SVBuffer[48];
    static Vertex *VA[3];

    Texture Tx;
    Tx._12xRes = 6;
    Tx._12yRes = 6;
    Tx._stateFlags = 0;
    Tx._PF = PF_R5G6B5;
#ifdef _FLY_OVER_WIN_CE
    // Set to Eyal's Texture mapper.
    RR_SetTextureMapper(TMMapper_Interface);
    Tx._stateFlags |= TSF_BLOCKTILED;
#else
    // Set to Refraster with std. inner
    RR_SetTextureMapper(TMMapper_Standard);
    RR_SetTextureInner(TInner_Standard);
#endif

    for(Q=Quads,QE=Q+NumQuads;Q<QE;Q++)
    {
        Vertex *SVBWrite = SVBuffer;
        int was_visible = 0;
        Tx._data = Q->Tex;
        RR_SetTexture(&Tx);
        FillColor_ = (byte)Q->drawlev;

        C_Flags = Vtx_Visibility&(Q->A.Flags|Q-
>B.Flags|Q->C.Flags|Q->D.Flags);
        g_lvldiff = Q->availlev;
    }
}

```

```

Vertex **Vtx = C_Buf;

//if ( GetAsyncKeyState('B') )
//{
/*      *Vtx++ = &Q->D;

      /// Add (Insert) Vertices to fix CRACKS

      long qlvl = Q->MT->Level;
      long qmaxlev = g_TM.MaxLevel - 1;
      long qscanline = (1 << qmaxlev) + 2;
      long qlength = 1 << (qmaxlev - qlvl);
      long qposx = qlength * Q->MT->TX + 1;
      long qposy = qlength * Q->MT->TY + 1;

      byte * qscantop = g_TM.ExpansionMap + qposy *
qscanline + qposx;
      byte * qscanbottom = qscantop + qlength *
qscanline;
      const long maxlev = g_TM.MaxLevel - 1;
      float t;

      // SCAN D->C
      byte * qscan = qscanbottom,
            * qscanend = qscanbottom+qlength;

      byte read;
      // Initial READ from expansion map
      read = *qscan;
      qscan += 1 << (maxlev - read);
      t = powerbase2[qlvl - read];

      while (qscan<qscanend)
      {
          // Subsequent READS / VERTEX SPLICING
          SpliceVertex(&Q->D,&Q->C,t,SVBWrite);
          *Vtx++ = SVBWrite++;
          read = *qscan;
          qscan += 1 << (maxlev - read);
          t += powerbase2[qlvl - read];
      }

      *Vtx++ = &Q->C;

      // SCAN C->B
      qscan = qscanbottom+qlength-qscanline;

```

```

qscanend = qscantop+qlength;
read = *qscan;
qscan -= qscanline << (maxlev - read);
t = powerbase2[qlvl - read];
while (qscan>=qscanend)
{
    // Subsequent READS / VERTEX SPLICING
    SpliceVertex(&Q->C,&Q->B,t,SVBWrite);
    *Vtx++ = SVBWrite++;
    read = *qscan;
    qscan -= qscanline << (maxlev - read);
    t += powerbase2[qlvl - read];
}

*Vtx++ = &Q->B;

// SCAN B->A
qscan = qscantop-qscanline+qlength-1;
qscanend = qscantop-qscanline;
read = *qscan;
qscan -= 1 << (maxlev - read);
t = powerbase2[qlvl - read];
while (qscan>=qscanend)
{
    SpliceVertex(&Q->B,&Q->A,t,SVBWrite);
    *Vtx++ = SVBWrite++;
    read = *qscan;
    qscan -= 1 << (maxlev - read);
    t += powerbase2[qlvl - read];
}

*Vtx++ = &Q->A;

// SCAN A->D
qscan = qscantop-1;
qscanend = qscanbottom-1;
read = *qscan;
qscan += qscanline << (maxlev - read);
t = powerbase2[qlvl - read];
while (qscan<qscanend)
{
    SpliceVertex(&Q->A,&Q->D,t,SVBWrite);
    *Vtx++ = SVBWrite++;
    read = *qscan;
    qscan += qscanline << (maxlev - read);
    t += powerbase2[qlvl - read];
}*/

```

```

    //} else {
        *Vtx++ = &Q->D;
        *Vtx++ = &Q->C;
        *Vtx++ = &Q->B;
        *Vtx++ = &Q->A;
    //}

    C_NumVtx = Vtx - C_Buf;

    if (C_Flags)
    {
        FrustrumClipper(Referance_Rasterizer);
    }
    else
    {
        C_Prim = C_Buf;
        MT = Q->MT;
        while (MT->Level>Q->availlev) MT=MT->Parent;
        if (0)//(MT->Flags & MT_JIT)
        {
            //MT->Flags &= ~MT_JIT;
            i = C_NumVtx;
            C_NumVtx = 3;
            VA[0] = C_Prim[0];
            VA[2] = C_Prim[1];
            for(j=2;j<i;j++)
            {
                VA[1] = VA[2];
                VA[2] = C_Prim[j];
                //Texture_Mapper(&F,C_Buf);
                Rasterizer(VA);
            }
        }
        else {
            Referance_Rasterizer(C_Prim);
        }
    }

    if (C_NumVtx) was_visible=1;

    /*C_Buf[0] = F.A = &Q->D;
    C_Buf[1] = F.B = &Q->B;
    C_Buf[2] = F.C = &Q->A;

    C_Flags = Vtx_Visibility&(Q->A.Flags|Q->B.Flags|Q->D.Flags);
    C_NumVtx = 3;

```

```

        if (C_Flags)
            FrustrumClipper(&F);
        else
            Rasterizer(&F,C_Buf);
            Texture_Mapper(&F,C_Buf);

        if (C_NumVtx) was_visible=1;*/

        if (was_visible)
        {
            // Reinsert to Cache
            MT = Q->MT;
            while (MT->Level>Q->availlev) MT=MT->Parent;
            for(;MT;MT=MT->Parent)
                if (MT->Texture)
                    UpdateTNode(MT->Texture);
                else
                    int bummer = 1;

            // Debugging code for distorted polygon
            surface

            /*float surface = PolygonSurface();
            static float bound = 1.0;
            if (GetAsyncKeyState('V')) bound*=1.02;
            if (GetAsyncKeyState('C')) bound/=1.02;
            if (surface > bound) {
                VA[0] = C_Prim[0];
                VA[2] = C_Prim[1];

                for(i=2;i<C_NumVtx;++i)
                {
                    VA[1] = VA[2];
                    VA[2] = C_Prim[i];
                    Rasterizer(&F,VA);
                }
            }
            }else {
                int yadda = 1;
            }*/
            // for visible polygons, look for the
lowest-level parent
            // missing a texture, and add a download
request for that parent.
            if (g_TM.UpdateDL)
            {
                MT = Q->MT;
                if (MT->Texture) continue;

```

```

                                while(!MT->Parent->Texture) MT=MT-
>Parent;
                                DLAdd(Q,MT);
                                }
                                }
                                }

```

// renders the contents (tiles) inside TM's quadtree

```

void TileManagerRender()
{
    if (g_TM.UpdatedDL) DLReset();
    QuadReset();
    TileRecalcPlane();

    // generate square
    Vector3 A,B,C,D;
    VectorCopy(&A,&g_TM.Origin);
    VectorLComb(&A,&g_TM.XAxis,1.0f,(float)g_TM.TSize[0],&B
);
    VectorLComb(&A,&g_TM.YAxis,1.0f,(float)g_TM.TSize[0],&D
);
    VectorLComb(&B,&g_TM.YAxis,1.0f,(float)g_TM.TSize[0],&C
);

    Vector3 *VP[4] = {&A,&B,&C,&D};
    Fast_WithinFrustrum_Init(&g_VP,VP);
    CalculatedDDX(VP);

    // niark - clear expansion map
    dword EMLen = (1 << (g_TM.MaxLevel-1)) +2;
    memset(g_TM.ExpansionMap,0,sizeof(byte)* EMLen*EMLen
);

    g_TM.DataIntegrity = g_TM.DataIntegrityNorm =
g_TM.DataIntegrityMax = 0.0f;
    DrawTileMap(g_TM.MT,VP[0],VP[1],VP[2],VP[3]);
    g_TM.DataIntegrity = 1.0f - log(g_TM.DataIntegrity /
g_TM.DataIntegrityMax) /
(float)log(g_TM.DataIntegrityNorm/g_TM.DataIntegrityMax);
    RenderQuads();

    /* need only for the debugging- create the little blue
square
    // understand it.
    // render expansion map

```

```

_/*long i,j;
word *ptr = g_pVPage+100;
byte *src = g_TM.ExpansionMap;
for(j=0;j<EMLen;j++)
{
    for(i=0;i<EMLen;i++)
    {
        *ptr++ = Palette[*src++];
    }
    ptr += g_xRes-EMLen;
}*/

//TileMapRemoveJIT();
TTexStorage2Cache();

static long LastUpdateTime;
if (g_TM.UpdateDL)
{
    LastUpdateTime = g_timer;
    DLManager();
}
// 1 second delay
g_TM.UpdateDL = (g_timer - LastUpdateTime > TMR_RESO);
}

```

Module: DL.cpp

---

```
dword DLGetNext(DownloadRequest *pRet)
{
    //Lock DTL MUTEX
    WaitForSingleObject(g_hDataMutex, INFINITE);

    DownloadRequest *Retval = 0;

    for(;;)
    {
        if (DTLNextAvail < DTLMax)
        {
            // check this map tile isn't already being
downloaded,
            Retval = DTL+DTLNextAvail++;
            if (Retval->MT->Flags & MT_Downloading)
            {
                Retval = 0;
                continue;
            }
            Retval->MT->Flags |= MT_Downloading;
        }

        break;
    }

    if (Retval)
        memcpy(pRet, Retval, sizeof(DownloadRequest));

    //Unlock DTL MUTEX
    ReleaseMutex(g_hDataMutex);

    return (dword)Retval;
}

void DLManager()
{
    static DownloadRequest **DPL = NULL;
    static DownloadRequest **DPS = NULL;

    static dword I;

    dword N = g_TM.DLNumRequests;

    if (N > DTLSize)
    {
```



```

        // re-allocate DPL/DPS and downloader thread's
list
        DPL = (DownloadRequest
**)realloc(DPL,sizeof(DownloadRequest *) *N);
        DPS = (DownloadRequest
**)realloc(DPS,sizeof(DownloadRequest *) *N);

        // note. information inside DTL still valid
because
        // the [partial] list contents were transferred
during
        // the reallocation

        // get DTL MUTEX
        WaitForSingleObject(g_hDataMutex, INFINITE);

        // realloc DTL
        DTL = (DownloadRequest
*)realloc(DTL,sizeof(DownloadRequest)*N);
        DTLSize = N;

        // release DTL MUTEX
        //ReleaseMutex(g_hDataMutex);
    }
    // rebuild DPL
    for(I=0;I<N;I++)
    {
        DPL[I] = g_TM.DLRequests+I;
    }

    // sort into DPL (DPS may be usable as temporary
storage)
    PrioritySort(DPL,DPS,N);

    /*Quad *Q,*QE;
    if (GetAsyncKeyState('U'))
    {
        for(Q=Quads,QE=Q+NumQuads;Q<QE;Q++)
        {
            if (Q->MT == DPL[0]->MT)
            {
                long x = Q->A.PX;
                long y = Q->A.PY;
                if (x>0&&x<g_xRes-1&&y>0&&y<g_yRes-1)
                {
                    word *w = g_pVPage+y*g_xRes+x;

```

```

        w[-(sdword)g_xRes] = w[-1] = w[0]
    = w[1] = w[g_xRes] = 0xFFFF;
    }
    }
}*/

// get DTL MUTEX
//WaitForSingleObject(g_hDataMutex, INFINITE);

// transfer DPS[I] to a new block inside DTL
for(I=0;I<N;I++)
    memcpy(DTL+I,DPL[I],sizeof(DownloadRequest));
//memcpy(DTL,g_TM.DLRequests,sizeof(DownloadRequest)*N
);

DTLMax = N;
// reset DTL Next Available number to 0
DTLNextAvail = 0;

// release DTL MUTEX
ReleaseMutex(g_hDataMutex);
}

void DLDownloaderThread(void *var)
{
    DownloadRequest    dl;

    // increase downloader thread count
//    nThreads = GetNThreads();
//    SetNThreads(nThreads+1);

    l_nDownloaderThreads++;

    for (;;)
    {
        while (DLGetNext(&dl) == 0)
        {
            // need we terminate ?
            if (l_stopDownloadThreads)
            {
                // yes, break loop
                break;
            }

            // sleep some time, then check again
            Sleep(100);
        }
    }
}

```

```

?      // do we need to terminate this thread and others

      if (l_stopDownloadThreads)
      {
          // decrease number of running threads
          l_nDownloaderThreads--;

          return;
      }

      byte *buf;
      dword len;

      Http11Request( g_TM.pServerName,
                     g_TM.pMapDir,
                     dl.FOffset,

      dl.FOffset+g_TM.IMAGE_CELL_SIZE-1,
                     g_TM.serverPort,
                     (void**)&buf, &len);

      //Sleep(500); // forcefully delays downloading,
used for debugging
      if (len)
          DLRecieve(dl.MT,buf);
      else
          dl.MT->Flags &=~MT_Downloading;

      delete buf;
  }
}

```

Module: DCam.cpp

---

```
void Dynamic_Camera(DCam *DC)
{
#ifdef _FLY_OVER_WIN_CE
    Camera *Cm = DC->Cm;
    Vector3 V;
    Matrix R,M;

    float dt = g_fgdttime;

    if (GetAsyncKeyState('I')) DC->KS*=expf(dt*2.0f);
    if (GetAsyncKeyState('O')) DC->KS*=expf(-dt*2.0f);

    // Kinematics
    VectorScale(&DC->Vel,dt,&V);
    VectorSelfAdd(&Cm->Position,&V);
    VectorScale(&DC->RVel,dt,&V);
    VectorSelfAdd(&DC->Rotation,&V);
    // rotation clipping
    //if (DC->Rotation.x > HALFPI*0.999) DC->Rotation.x =
HALFPI*0.999;
    //if (DC->Rotation.x <-HALFPI*0.999) DC->Rotation.x ==
HALFPI*0.999;
    if (DC->Rotation.y <-PI) DC->Rotation.y += TWOPI;
    if (DC->Rotation.y > PI) DC->Rotation.y -= TWOPI;

    // Orientation
    //VectorScale(&DC->RVel,dt,&V);
    //Euler_Angles(R,V.x,V.y,V.z);
    //MatrixXMatrix(R,DC->VM,M);
    Matrix BT,ET;
    Euler_Angles(BT,0,DC->Rotation.y,0);
    Euler_Angles(ET,DC->Rotation.x,0,0);
    MatrixXMatrix(ET,BT,DC->VM);
    // MatrixCopy(DC->VM,M);
    // Euler_Angles(R,0,0,DC->Roll);
    // MatrixXMatrix(R,DC->VM,Cm->VM);

    // Adjust by Keyboard Input
    float dtks = dt * DC->KS;
    if (GetAsyncKeyState(VK_NEXT))
    {
        DC->Vel.x += dtks * Cm->VM[0];
        DC->Vel.y += dtks * Cm->VM[1];
        DC->Vel.z += dtks * Cm->VM[2];
    }
}
```

```

    }
    if (GetAsyncKeyState(VK_END))
    {
        DC->Vel.x -= dtks * Cm->VM[0];
        DC->Vel.y -= dtks * Cm->VM[1];
        DC->Vel.z -= dtks * Cm->VM[2];
    }
    if (GetAsyncKeyState(VK_SUBTRACT))
    {
        //DC->Vel.x += dtks * Cm->VM[3];
        DC->Vel.y += dtks; // * Cm->VM[4];
        //DC->Vel.z += dtks * Cm->VM[5];
    }
    if (GetAsyncKeyState(VK_ADD))
    {
        //DC->Vel.x -= dtks * Cm->VM[3];
        DC->Vel.y -= dtks; // * Cm->VM[4];
        //DC->Vel.z -= dtks * Cm->VM[5];
    }
    /*f (GetAsyncKeyState('A'))
    {
        DC->Vel.x += dtks * DC->VM[6];
        DC->Vel.y += dtks * Cm->VM[7];
        DC->Vel.z += dtks * DC->VM[8];
    }
    if (GetAsyncKeyState('Z'))
    {
        DC->Vel.x -= dtks * DC->VM[6];
        DC->Vel.y -= dtks * Cm->VM[7];
        DC->Vel.z -= dtks * DC->VM[8];
    }*/
    Vector3 Z_Motion = {DC->VM[6],0,DC->VM[8]}; // Z
motion
    float z1 = VectorLength(&Z_Motion);
    if (z1<0.7)
    {
        VectorMake(&Z_Motion,BT[6],0,BT[8]);
    } else {
        VectorNorm(&Z_Motion);
    }

    if (GetAsyncKeyState('A'))
    {
        DC->Vel.x += dtks * Z_Motion.x;
        DC->Vel.z += dtks * Z_Motion.z;
    }
    if (GetAsyncKeyState('Z'))

```

```

{
    DC->Vel.x -= dtks * Z_Motion.x;
    DC->Vel.z -= dtks * Z_Motion.z;
}
if (GetAsyncKeyState('S'))
{
    DC->Vel.x += dtks * DC->VM[6];
    DC->Vel.y += dtks * DC->VM[7];
    DC->Vel.z += dtks * DC->VM[8];
}
if (GetAsyncKeyState('X'))
{
    DC->Vel.x -= dtks * DC->VM[6];
    DC->Vel.y -= dtks * DC->VM[7];
    DC->Vel.z -= dtks * DC->VM[8];
}

// TODO - check if our window is active

float dtrks = dt * DC->RKS;
if (GetAsyncKeyState(VK_UP)) DC->RVel.x += dtrks;
if (GetAsyncKeyState(VK_DOWN)) DC->RVel.x -= dtrks;

if (GetAsyncKeyState(VK_RIGHT))
{
    //DC->Roll += dt * DC->TRoll;
    DC->RVel.y += dtrks;
}
if (GetAsyncKeyState(VK_LEFT))
{
    //DC->Roll -= dt * DC->TRoll;
    DC->RVel.y -= dtrks;
}
if (GetAsyncKeyState(VK_HOME)) DC->RVel.z += dtrks;
if (GetAsyncKeyState(VK_PRIOR)) DC->RVel.z -= dtrks;

// Movement dampening
float falloff;
falloff = (float)exp(-dt*DC->Damp);
VectorSelfScale(&DC->Vel, falloff);
// Rotation dampening
falloff = (float)exp(-dt*DC->RDamp);
VectorSelfScale(&DC->RVel, falloff);

// Roll dampening
//falloff = (float)exp(-dt*DC->RlDamp);

```

```

//DC->Roll *= falloff;
DC->Roll += DC->RVel.z * 0.05;

// Verify height
if (Cm->Position.y < DC->minY)
    Cm->Position.y = DC->minY;

// Auto-Levelling
Vector3 *VMP = (Vector3 *)DC->VM;
VMP[0].y = 0.0;
VectorNorm(VMP);
VectorCross(VMP+2, VMP, VMP+1);
VectorNorm(VMP+1);
VectorCross(VMP, VMP+1, VMP+2);

//MatrixIdentity(Cm->VM);
Euler_Angles(R, 0, 0, DC->Roll);
MatrixXMatrix(R, DC->VM, Cm->VM);

#else
Camera *Cm = DC->Cm;
Vector3 V;
Matrix R, M;

float dt = g_fgdttime;

if (GetAsyncKeyState('I')) DC->KS*=expf(dt*2.0f);
if (GetAsyncKeyState('O')) DC->KS*=expf(-dt*2.0f);

// Kinematics
VectorScale(&DC->Vel, dt, &V);
VectorSelfAdd(&Cm->Position, &V);
VectorScale(&DC->RVel, dt, &V);
VectorSelfAdd(&DC->Rotation, &V);
// rotation clipping
//if (DC->Rotation.x > HALFPI*0.999) DC->Rotation.x =
HALFPI*0.999;
//if (DC->Rotation.x <-HALFPI*0.999) DC->Rotation.x ==
HALFPI*0.999;
if (DC->Rotation.y <-PI) DC->Rotation.y += TWOPI;
if (DC->Rotation.y > PI) DC->Rotation.y -= TWOPI;

// Orientation
//VectorScale(&DC->RVel, dt, &V);
//Euler_Angles(R, V.x, V.y, V.z);
//MatrixXMatrix(R, DC->VM, M);
Matrix BT, ET;

```

```

    Euler_Angles(BT,0,DC->Rotation.y,0);
    Euler_Angles(ET,DC->Rotation.x,0,0);
    MatrixXMatrix(ET,BT,DC->VM);
//    MatrixCopy(DC->VM,M);
//    Euler_Angles(R,0,0,DC->Roll);
//    MatrixXMatrix(R,DC->VM,Cm->VM);

    // Adjust by Keyboard Input
    float dtks = dt * DC->KS;
/*    if (GetAsyncKeyState(VK_NEXT))
    {
        DC->Vel.x += dtks * Cm->VM[0];
        DC->Vel.y += dtks * Cm->VM[1];
        DC->Vel.z += dtks * Cm->VM[2];
    }
    if (GetAsyncKeyState(VK_END))
    {
        DC->Vel.x -= dtks * Cm->VM[0];
        DC->Vel.y -= dtks * Cm->VM[1];
        DC->Vel.z -= dtks * Cm->VM[2];
    }
    if (GetAsyncKeyState(VK_SUBTRACT))
    {
        //DC->Vel.x += dtks * Cm->VM[3];
        DC->Vel.y += dtks; // * Cm->VM[4];
        //DC->Vel.z += dtks * Cm->VM[5];
    }
    if (GetAsyncKeyState(VK_ADD))
    {
        //DC->Vel.x -= dtks * Cm->VM[3];
        DC->Vel.y -= dtks; // * Cm->VM[4];
        //DC->Vel.z -= dtks * Cm->VM[5];
    }
    f (GetAsyncKeyState('A'))
    {
        DC->Vel.x += dtks * DC->VM[6];
        DC->Vel.y += dtks * Cm->VM[7];
        DC->Vel.z += dtks * DC->VM[8];
    }
    if (GetAsyncKeyState('Z'))
    {
        DC->Vel.x -= dtks * DC->VM[6];
        DC->Vel.y -= dtks * Cm->VM[7];
        DC->Vel.z -= dtks * DC->VM[8];
    }
*/
    Vector3 Z_Motion = {DC->VM[6],0,DC->VM[8]}; // Z
motion

```



```

float z1 = VectorLength(&Z_Motion);
if (z1<0.000001)
{
    VectorMake(&Z_Motion,-DC->VM[3],-DC->VM[4],-DC-
>VM[5]);
} else {
    VectorNorm(&Z_Motion);
}

static int state = 0;
static int pressed = 0;

if ( GetAsyncKeyState(VK_F23) && (!pressed) )
{
    pressed = 1;
    state = !state;
}
else
{
    pressed = 0;
}

float dtrks = dt * DC->RKS;

if (state)
{
    if (GetAsyncKeyState(VK_UP))
    {
        DC->Vel.x += dtrks * DC->VM[6];
        DC->Vel.y += dtrks * DC->VM[7];
        DC->Vel.z += dtrks * DC->VM[8];
    }
    if (GetAsyncKeyState(VK_DOWN))
    {
        DC->Vel.x -= dtrks * DC->VM[6];
        DC->Vel.y -= dtrks * DC->VM[7];
        DC->Vel.z -= dtrks * DC->VM[8];
    }
    if (GetAsyncKeyState(VK_RIGHT))
    {
        DC->RVel.x += dtrks;
    }
    if (GetAsyncKeyState(VK_LEFT))
    {
        DC->RVel.x -= dtrks;
    }
}

```

```

else
{
    if (GetAsyncKeyState(VK_UP))
    {
        DC->Vel.x += dtks * Z_Motion.x;
        DC->Vel.z += dtks * Z_Motion.z;
    }
    if (GetAsyncKeyState(VK_DOWN))
    {
        DC->Vel.x -= dtks * Z_Motion.x;
        DC->Vel.z -= dtks * Z_Motion.z;
    }
    if (GetAsyncKeyState(VK_RIGHT))
    {
        //DC->Roll += dt * DC->TRoll;
        DC->RVel.y += dtrks;
    }
    if (GetAsyncKeyState(VK_LEFT))
    {
        //DC->Roll -= dt * DC->TRoll;
        DC->RVel.y -= dtrks;
    }
}
/* if (GetAsyncKeyState('S'))
{
    DC->Vel.x += dtks * DC->VM[6];
    DC->Vel.y += dtks * DC->VM[7];
    DC->Vel.z += dtks * DC->VM[8];
}
if (GetAsyncKeyState('X'))
{
    DC->Vel.x -= dtks * DC->VM[6];
    DC->Vel.y -= dtks * DC->VM[7];
    DC->Vel.z -= dtks * DC->VM[8];
}*/

// TODO - check if our window is active

// if (GetAsyncKeyState(VK_UP)) DC->RVel.x += dtrks;
// if (GetAsyncKeyState(VK_DOWN)) DC->RVel.x -= dtrks;

// if (GetAsyncKeyState(VK_HOME)) DC->RVel.z += dtrks;
// if (GetAsyncKeyState(VK_PRIOR)) DC->RVel.z -= dtrks;

// Movement dampening
float falloff;

```

```

    falloff = (float)exp(-dt*DC->Damp);
    VectorSelfScale(&DC->Vel,falloff);
    // Rotation dampening
    falloff = (float)exp(-dt*DC->RDamp);
    VectorSelfScale(&DC->RVel,falloff);

    // Roll dampening
    //falloff = (float)exp(-dt*DC->RlDamp);
    //DC->Roll *= falloff;
    DC->Roll += DC->RVel.z * 0.05;

    // Verify height
    if (Cm->Position.y < DC->minY)
        Cm->Position.y = DC->minY;

    // Auto-Levelling
    Vector3 *VMP = (Vector3 *)DC->VM;
    VMP[0].y = 0.0;
    VectorNorm(VMP);
    VectorCross(VMP+2,VMP,VMP+1);
    VectorNorm(VMP+1);
    VectorCross(VMP,VMP+1,VMP+2);

    //MatrixIdentity(Cm->VM);
    Euler_Angles(R,0,0,DC->Roll);
    MatrixXMatrix(R,DC->VM,Cm->VM);
#endif
}

// funcs
dword Http11Request(char *pServer, // 'www.imr.org'
                    char *pFile, //
                    'images/logo.jpg'
                    dword rangeFrom, // 450, -1
                    (-1 == from begining of file)
                    dword rangeTo, // 500, -
                    1 (-1 == till end of file)
                    word port, // 80
                    void **ppDestFileBuf,
                    dword *pDestFileLen)
{
    struct hostent *pHost;
    struct sockaddr_in serverAddr;
    SOCKET serverSock;
    int retVal;
    dword exitError = 0;

```

```

char                *recipientServerName;

// nulify reply
*ppDestFileBuf = 0;
*pDestFileLen = 0;

// are using a proxy server ?
recipientServerName = l_useProxy ? l_pProxyServer :
pServer;

// retrieve server IP
if (isalpha(recipientServerName[0]))
    pHost = gethostbyname(recipientServerName);
else
{
    dword addr;
    addr = inet_addr(recipientServerName);
    pHost = gethostbyaddr((char *)&addr, 4, AF_INET);
}

if (pHost == 0)
{
    /*LogMsg("Http11Request(): unable to resolve
[%s], Error [%d]",
        recipientServerName, WSAGetLastError());*/

    HTTP_ERROR(Http11Request_ErrLv11)
}

// Copy the resolved information into the sockaddr_in
structure
memset(&serverAddr, 0, sizeof(serverAddr));
memcpy(&(serverAddr.sin_addr), pHost->h_addr, pHost-
>h_length);
serverAddr.sin_family = pHost->h_addrtype;
serverAddr.sin_port = htons(l_useProxy ? l_proxyPort :
port);

// open a socket
if ((serverSock = socket(AF_INET, SOCK_STREAM, 0)) ==
INVALID_SOCKET)
{
    /*LogMsg("Http11Request(): failed to open socket
to server [%s], Error [%d]",
        pServer, WSAGetLastError());*/

    HTTP_ERROR(Http11Request_ErrLv11)
}

```

```

    }

    /// connect to the server
    if (connect(serverSock, (struct sockaddr*)&serverAddr,
sizeof(serverAddr))
        == SOCKET_ERROR)
    {
        //LogMsg("Http11Request(): failed to connect to
server [%s], Error [%d]",
        //davidi pServer, WSAGetLastError());

        HTTP_ERROR(Http11Request_ErrLvl2)
    }

    // generate our request
    char *pRequestBuf;

    pRequestBuf = new char [REQUEST_BUFFER_SIZE];

    memset(pRequestBuf, 0, REQUEST_BUFFER_SIZE);

    //GET /index.html HTTP/1.1
    //Connection: Keep-Alive
    //Host: 10.1.1.1
    //Range: bytes=10-20

    //GET http://www.yahoo.com/blah/index.html HTTP/1.0
    //Host: www.yahoo.com
    //Proxy-Connection: Keep-Alive

    strcpy(pRequestBuf, "GET ");

    if (!l_useProxy)
    {
        strcat(pRequestBuf, pFile);
        strcat(pRequestBuf, " HTTP/1.1\r\n");
        strcat(pRequestBuf, "Connection: Keep-
Alive\r\n");
    }
    else
    {
        strcat(pRequestBuf, "http://");
        strcat(pRequestBuf, pServer);
        strcat(pRequestBuf, pFile);
        strcat(pRequestBuf, " HTTP/1.0\r\n");
        strcat(pRequestBuf, "Proxy-Connection: Keep-
Alive\r\n");
    }

```

```

}

strcat(pRequestBuf, "Host: ");
strcat(pRequestBuf, pServer);
strcat(pRequestBuf, "\r\n");

// do we have byte ranges ?
if (rangeFrom != -1 || rangeTo != -1)
{
    // we need to take care of ranges
    strcat(pRequestBuf, "Range: bytes=");

    // from range
    char rangeString[32];

    sprintf(rangeString, "%d-", rangeFrom == -1 ? 0 :
rangeFrom);
    strcat(pRequestBuf, rangeString);

    // to range
    if (rangeTo != -1)
    {
        sprintf(rangeString, "%d", rangeTo);
        strcat(pRequestBuf, rangeString);
    }

    strcat(pRequestBuf, "\r\n");
}

strcat(pRequestBuf, "\r\n");

// send request
if (send(serverSock, pRequestBuf, strlen(pRequestBuf),
0) == SOCKET_ERROR)
{
    /*LogMsg("Http11Request(): send() failed, Error
[%d]",
        pServer, WSAGetLastError());*/

    delete pRequestBuf;

    HTTP_ERROR(Http11Request_ErrLvl2)
}

delete pRequestBuf;

```

```

// receive response

//////////
// NOTE! response buffer now gets padded with
additional 2Kbytes due to out of array bound
// writes made by the following code. This is a bug
detected by Yoni on 27/7/01,
// however it isn't fixed because of my inexperience
with networking code.
char *pResponseBuf;

//pResponseBuf = new char[RESPONSE_BUFFER_SIZE];
char *paddedBuffer;
paddedBuffer = new char [RESPONSE_BUFFER_SIZE+2048];
pResponseBuf = &paddedBuffer[1024];

if ((retVal = recv(serverSock, (char*)pResponseBuf,
RESPONSE_BUFFER_SIZE, 0)) <= 0)
{
    /*LogMsg("Http11Request(): recv() failed,%sError
[%d]",
            (retVal == 0) ? " connection gracefully
closed, " : " ",
            WSAGetLastError());*/

    HTTP_ERROR(Http11Request_ErrLv13)
}

pResponseBuf[retVal] = 0;

// fetch http return-code
// 'HTTP/1.1 206 Partial Content'

dword    httpVer[2], httpResponseCode;

sscanf((const char*)pResponseBuf, "HTTP/%d.%d %d",
&httpVer[0], &httpVer[1], &httpResponseCode);

switch (httpResponseCode)
{
case 200: // ok
case 206: // partial content
    break;

case 302: // moved temporarily
    break;

```

```

        case 204: // no content
            //LogMsg("Http11Request(): HTTP error 204 (no
content)");
            HTTP_ERROR(Http11Request_ErrLvl3)

        case 400: // bad request
            //LogMsg("Http11Request(): HTTP error 400 (bad
request)");
            HTTP_ERROR(Http11Request_ErrLvl3)

        case 401: // unauthorized
            //LogMsg("Http11Request(): HTTP error 401
(unauthorized)");
            HTTP_ERROR(Http11Request_ErrLvl3)

        case 404: // not found
            //LogMsg("Http11Request(): HTTP error 404 (not
found)");
            HTTP_ERROR(Http11Request_ErrLvl3)

        case 405: // method not allowed
            //LogMsg("Http11Request(): HTTP error 405 (method
not allowed)");
            HTTP_ERROR(Http11Request_ErrLvl3)

        case 406: // not acceptable
            //LogMsg("Http11Request(): HTTP error 406 (not
acceptable)");
            HTTP_ERROR(Http11Request_ErrLvl3)

        case 500: // server error
            //LogMsg("Http11Request(): HTTP error 503 (server
error)");
            HTTP_ERROR(Http11Request_ErrLvl3)

        case 503: // out of resources
            //LogMsg("Http11Request(): HTTP error 503 (out of
resources)");
            HTTP_ERROR(Http11Request_ErrLvl3)

        case 505: // http version not supported
            //LogMsg("Http11Request(): HTTP error 505 (http
version not supported)");
            HTTP_ERROR(Http11Request_ErrLvl3)

        default:

```



```

        //LogMsg("Http11Request(): HTTP error %d",
httpResponseCode);
        HTTP_ERROR(Http11Request_ErrLvl3)
    }

    // if we reached here, it means the file is coming

    dword        receivedFileLen, receivedFileOfs;
    byte *pReceivedFile;
    dword        expectedFileSize;

    receivedFileLen = 0;
    receivedFileOfs = 0;
    expectedFileSize = 0;
    pReceivedFile = 0;

    // do we know the file size ?

    if (rangeTo != -1)
    {
        // if we have a 'to' range, it means we can
determine the requested
        // data length

        if (rangeFrom == -1)
            expectedFileSize = rangeTo;
        else
            expectedFileSize = rangeTo - rangeFrom + 1;
    }
    else
        // look for content-length
    {
        char *pClPos;

        pClPos = strstr(pResponseBuf, "Content-Length:
");

        if (pClPos)
            sscanf(pClPos, "Content-Length: %d",
&expectedFileSize);
    }

    // did we manage to devise a content-length ?
    receivedFileLen = expectedFileSize ? expectedFileSize
: RESPONSE_BUFFER_SIZE;

    pReceivedFile = (byte*)malloc(receivedFileLen);

```

```

    // find double crlf in the 1'st packet, this tells us
where the header
    // ends and where the data begins

    char *dblCRLF;

    while ((dblCRLF = strstr(pResponseBuf, "\r\n\r\n")) ==
0)
    {
        // odd, this packet doesn't contain our signal
        {
            // clear buffer
            memset(pResponseBuf, 0, RESPONSE_BUFFER_SIZE);

            // grab packet
            if ((retVal = recv(serverSock,
(char*)pResponseBuf, RESPONSE_BUFFER_SIZE, 0)) <= 0)
            {
                /*LogMsg("Http11Request(): recv()
failed,%sError [%d]",
                    (retVal == 0) ? " connection gracefully
closed, " : " ",
                    WSAGetLastError());*/

                HTTP_ERROR(Http11Request_ErrLvl4)
            }
        }

        // is there any binary data for us in this packet ?
        dword crlfPos;

        crlfPos = (dword)(dblCRLF - pResponseBuf);

        if (crlfPos+4 < (dword)retVal)
        {
            // there's binary data in the 1'st packet
            // since our target buffer is the size of the
incoming buffer
            // and the incoming buffer includes a header, we
can safely copy
            // without fear of overlapping

            dword dataSize;

            dataSize = retVal - crlfPos - 4;

```

```

        memcpy(pReceivedFile, pResponseBuf+crlfPos+4,
dataSize);

        receivedFileOfs = dataSize;
    }

    // did the entire file fit into the first packet ?
    if (!expectedFileSize || (receivedFileOfs <
expectedFileSize))
    {
        // no, get more packets

        // now we go into a loop of receiveing and adding
to our file
        while ((retVal = recv(serverSock,
(char*)pResponseBuf, RESPONSE_BUFFER_SIZE, 0)) != 0)
        {
            // did the latest 'recv' call end with a
socket error ?
            if (retVal == -1)
            {
                //davidi LogMsg("Http11Request():
recv() failed, Error [%d]", WSAGetLastError());

                HTTP_ERROR(Http11Request_ErrLvl4)
            }

            // add packet to downloaded file
            if (receivedFileOfs + retVal >=
receivedFileLen)
            {
                pReceivedFile =
(byte*)realloc(pReceivedFile,
receivedFileLen+RESPONSE_BUFFER_SIZE);

                receivedFileLen +=
RESPONSE_BUFFER_SIZE;
            }

            memcpy(pReceivedFile+receivedFileOfs,
pResponseBuf, retVal);

            receivedFileOfs += retVal;

            // did we get the entire file ?
            // no need to test for validity of
'expectedFileSize' since 'receivedFileOfs'

```

```

        // can't be 0
        if (receivedFileOfs == expectedFileSize)
            // yes, get out of the 'while' loop
            break;
    }
}

// is it possible connection was closed gracefully,
and we didn't receive the entire file ?
// if we have an expected size we can check this, but
i don't think it can happen
if (expectedFileSize && (receivedFileOfs <
expectedFileSize))
{
    //LogMsg("Http11Request(): recv() failed,
connection closed gracefully, but file not completely
received, Error [%d]", WSAGetLastError());

    HTTP_ERROR(Http11Request_ErrLvl4)
}

// here we are, connection was closed meaning we have
the file
*ppDestFileBuf = new byte[receivedFileOfs];
memcpy(*ppDestFileBuf, pReceivedFile,
receivedFileOfs);
*pDestFileLen = receivedFileOfs;

Http11Request_ErrLvl4:
    free(pReceivedFile);

Http11Request_ErrLvl3:
    //delete [] pResponseBuf;
    delete [] paddedBuffer;

Http11Request_ErrLvl2:
    closesocket(serverSock);

Http11Request_ErrLvl1:
    return exitError ? IMR_ERR : IMR_COOL;
}

static void FClipper_Near()
{
    dword I,J = 0;
    IB_ = C_Prim[0];

```

```

C_NFree = C_Isect;
for(I=0;I<C_NumVtx;I++)
{
    IA_ = IB_;
    IB_ = C_Prim[I+1];
    if (IA_->Flags & Vtx_Near)
    {
        if (IB_->Flags & Vtx_Near) continue;
    } else {
        C_Scnd[J++] = IA_;
        if (!(IB_->Flags & Vtx_Near)) continue;
    }
    IVal_ = (g_VP.ZN-IA_->TPos.z)/(IB_->TPos.z-IA_-
>TPos.z);
    INVal_ = 1.0f-IVal_;
    C_NFree->TPos.x = INVal_ * IA_->TPos.x + IVal_ *
IB_->TPos.x;
    C_NFree->TPos.y = INVal_ * IA_->TPos.y + IVal_ *
IB_->TPos.y;
    C_NFree->TPos.z = g_VP.ZN;
    // C_NFree->R = INVal_ * IA_->R + IVal_ * IB_->R;
    // C_NFree->G = INVal_ * IA_->G + IVal_ * IB_->G;
    // C_NFree->B = INVal_ * IA_->B + IVal_ * IB_->B;
    C_NFree->U = INVal_ * IA_->U + IVal_ * IB_->U;
    C_NFree->V = INVal_ * IA_->V + IVal_ * IB_->V;

    C_NFree->PX = C_NFree->TPos.x * C_rZN;
    C_NFree->PY = C_NFree->TPos.y * C_rZN;
    C_NFree->RZ = C_rZN;
    C_NFree->UZ = C_NFree->U * C_rZN;
    C_NFree->VZ = C_NFree->V * C_rZN;

    Calc_Flags(C_NFree);
    C_Scnd[J++] = C_NFree++;
}

C_NumVtx = J;
Clipper_Swap();
}

static void FClipper_Far()
{
    dword I,J = 0;
    IB_ = C_Prim[0];

    C_NFree = C_Isect+2;
    for(I=0;I<C_NumVtx;I++)

```

```

{
    IA_ = IB_;
    IB_ = C_Prim[I+1];
    if (IA_>Flags & Vtx_Far)
    {
        if (IB_>Flags & Vtx_Far) continue;
    } else {
        C_Scnd[J++] = IA_;
        if (!(IB_>Flags & Vtx_Far)) continue;
    }
    IVal_ = (g_VP.ZF-IA_>TPos.z)/(IB_>TPos.z-IA_>TPos.z);
    INVal_ = 1.0f-IVal_;
    C_NFree->PX = (INVal_ * IA_>TPos.x + IVal_ *
IB_>TPos.x) * C_rZF;
    C_NFree->PY = (INVal_ * IA_>TPos.y + IVal_ *
IB_>TPos.y) * C_rZF;
    // C_NFree->R = INVal_ * IA_>R + IVal_ * IB_>R;
    // C_NFree->G = INVal_ * IA_>G + IVal_ * IB_>G;
    // C_NFree->B = INVal_ * IA_>B + IVal_ * IB_>B;
    C_NFree->RZ = C_rZF;
    C_NFree->UZ = C_rZF * (INVal_ * IA_>U + IVal_ *
IB_>U);
    C_NFree->VZ = C_rZF * (INVal_ * IA_>V + IVal_ *
IB_>V);
    Calc_Flags(C_NFree);
    C_Scnd[J++] = C_NFree++;
}

C_NumVtx = J;
Clipper_Swap();
}

// one day, with the appropriate technology, this will use
code-constructed procedures
static void FInterpolator()
{
    if (IVal_>1.0f || IVal_<0.0f)
    {
        int blat = 1;
    }
    C_NFree->PX = INVal_ * IA_>PX + IVal_ * IB_>PX;
    C_NFree->PY = INVal_ * IA_>PY + IVal_ * IB_>PY;
    // C_NFree->R = INVal_ * IA_>R + IVal_ * IB_>R;
    // C_NFree->G = INVal_ * IA_>G + IVal_ * IB_>G;
    // C_NFree->B = INVal_ * IA_>B + IVal_ * IB_>B;
    C_NFree->RZ = INVal_ * IA_>RZ + IVal_ * IB_>RZ;

```

```

C_NFree->UZ = INVal_ * IA_->UZ + IVal_ * IB_->UZ;
C_NFree->VZ = INVal_ * IA_->VZ + IVal_ * IB_->VZ;

// calculate (U,V)
float z = 1.0f/C_NFree->RZ;
C_NFree->U = C_NFree->UZ*z;
C_NFree->V = C_NFree->VZ*z;
}

static void FClipper_Left()
{
    dword I,J = 0;
    IB_ = C_Prim[0];

    C_NFree = C_Isect+4;
    for(I=0;I<C_NumVtx;I++)
    {
        IA_ = IB_;
        IB_ = C_Prim[I+1];
        if (IA_->Flags & Vtx_Left)
        {
            if (IB_->Flags & Vtx_Left) continue;
        } else {
            C_Scnd[J++] = IA_;
            if (!(IB_->Flags & Vtx_Left)) continue;
        }
        IVal_ = (g_VP.ClipX1-IA_->PX)/(IB_->PX-IA_->PX);
        INVal_ = 1.0f-IVal_;
        FInterpolator();
        C_NFree->PX = g_VP.ClipX1;
        Calc_YFlags(C_NFree);
        C_Scnd[J++] = C_NFree++;
    }

    C_NumVtx = J;
    Clipper_Swap();
}

static void FClipper_Right()
{
    dword I,J = 0;
    IB_ = C_Prim[0];

    C_NFree = C_Isect+6;
    for(I=0;I<C_NumVtx;I++)
    {
        IA_ = IB_;

```

```

        IB_ = C_Prim[I+1];
        if (IA_>Flags & Vtx_Right)
        {
            if (IB_>Flags & Vtx_Right) continue;
        } else {
            C_Scnd[J++] = IA_;
            if (!(IB_>Flags & Vtx_Right)) continue;
        }
        IVal_ = (g_VP.ClipX2-IA_>PX)/(IB_>PX-IA_>PX);
        INVal_ = 1.0f-IVal_;
        FInterpolator();
        C_NFree->PX = g_VP.ClipX2;
        Calc_YFlags(C_NFree);
        C_Scnd[J++] = C_NFree++;
    }

    C_NumVtx = J;
    Clipper_Swap();
}

static void FClipper_Up()
{
    dword I,J = 0;
    IB_ = C_Prim[0];

    C_NFree = C_Isect+8;
    for(I=0;I<C_NumVtx;I++)
    {
        IA_ = IB_;
        IB_ = C_Prim[I+1];
        if (IA_>Flags & Vtx_Up)
        {
            if (IB_>Flags & Vtx_Up) continue;
        } else {
            C_Scnd[J++] = IA_;
            if (!(IB_>Flags & Vtx_Up)) continue;
        }
        IVal_ = (g_VP.ClipY1-IA_>PY)/(IB_>PY-IA_>PY);
        INVal_ = 1.0f-IVal_;
        FInterpolator();
        C_NFree->PY = g_VP.ClipY1;
        C_Scnd[J++] = C_NFree++;
    }

    C_NumVtx = J;
    Clipper_Swap();
}

```



```

static void FClipper_Down()
{
    dword I,J = 0;
    IB_ = C_Prim[0];

    C_NFree = C_Isect+10;
    for(I=0;I<C_NumVtx;I++)
    {
        IA_ = IB_;
        IB_ = C_Prim[I+1];
        if (IA_>Flags & Vtx_Down)
        {
            if (IB_>Flags & Vtx_Down) continue;
        } else {
            C_Scnd[J++] = IA_;
            if (!(IB_>Flags & Vtx_Down)) continue;
        }
        IVal_ = (g_VP.ClipY2-IA_>PY)/(IB_>PY-IA_>PY);
        INVal_ = 1.0f-IVal_;
        FInterpolator();
        C_NFree->PY = g_VP.ClipY2;
        C_Scnd[J++] = C_NFree++;
    }

    C_NumVtx = J;
    Clipper_Swap();
}

void FrustrumClipper(void (*RasterFunc)(Vertex **VA))
{
    dword I;

    C_Buf[C_NumVtx] = C_Buf[0];
    C_Prim = C_Buf;
    C_Scnd = C_Buf2;

    if (C_Flags&Vtx_Near) FClipper_Near();
    if (C_Flags&Vtx_Far) FClipper_Far();
    if (C_Flags&Vtx_Left) FClipper_Left();
    if (C_Flags&Vtx_Right) FClipper_Right();
    if (C_Flags&Vtx_Up) FClipper_Up();
    if (C_Flags&Vtx_Down) FClipper_Down();

    /* if (!(CF->Txtr))

```

```

{1.04
    return;
}
//if
while (C_NumVtx==3)
{
    int mny = 0, ly = 2, ry = 1;
    if (C_Prim[mny]->PY > C_Prim[1]->PY)
    {
        mny = 1; ly = 0; ry = 2;
    }
    if (C_Prim[mny]->PY > C_Prim[2]->PY)
    {
        mny = 2; ly = 1; ry = 0;
    }

    if ((C_Prim[mny]->PY == C_Prim[ly]->PY) || (C_Prim[mny]->PY == C_Prim[ry]->PY)) return;

    if ( ((C_Prim[ry]->PX - C_Prim[mny]->PX) /
(C_Prim[ry]->PY - C_Prim[mny]->PY)) >
        ((C_Prim[ry]->PX - C_Prim[mny]->PX) /
(C_Prim[ry]->PY - C_Prim[mny]->PY)))
        test_texture(CF, C_Prim, C_NumVtx);
    C_NumVtx--;
}
C_NumVtx++;*/

/*if (C_NumVtx>3)
{
    C_Scnd[0] = C_Prim[0];
    C_Scnd[2] = C_Prim[1];
    dword J = C_NumVtx;
    C_NumVtx = 3;
    for(I=2;I<J;I++)
    {
        C_Scnd[1] = C_Scnd[2];
        C_Scnd[2] = C_Prim[I];

        //Rasterizer(CF,C_Scnd);
        Texture_Mapper(CF,C_Scnd);
    }
} else if (C_NumVtx==3)
{
    //Rasterizer(CF,C_Prim);
    Texture_Mapper(CF,C_Prim);
}
}

```

```

    */
if (0)//C_Flags & Vtx_Left)
{
    C_Scnd[0] = C_Prim[0];
    C_Scnd[2] = C_Prim[1];
    dword J = C_NumVtx;
    C_NumVtx = 3; // required for
    for(I=2;I<J;I++)
    {
        C_Scnd[1] = C_Scnd[2];
        C_Scnd[2] = C_Prim[I];

        //Rasterizer(CF,C_Scnd);
        //Texture_Mapper(C_Scnd);
    }
    C_NumVtx = J;
} else {
    if (C_NumVtx>=3 && RasterFunc)
        //Texture_Mapper(CF,C_Prim);
        RasterFunc(C_Prim);
}
}

```

Module: Window.cpp

---

```
// local variables
static byte          *l_pBibuf;
static BITMAPINFO    *l_pBi;
static HANDLE        l_WndThreadDone;
static byte          *l_pPage3 = 0;
static byte          l_pf;

#ifdef _FLY_OVER_WIN_CE
static HBITMAP        l_hBitmap = NULL;
static void           *l_pBits;
#endif

// global variables
HWND                  g_hWnd;
DWORD                 g_xRes, g_yRes, g_Pitch;
WORD                  *g_pVPage;

static byte IdentifyWindowPF()
{
#ifdef _FLY_OVER_WIN_CE

    struct MyBitmapInfo
    {
        BITMAPINFOHEADER bi;
        union
        {
            RGBQUAD colors[256];
            DWORD fields[256];
        };
    };
    MyBitmapInfo video;

    ZeroMemory(&video, sizeof(video));
    video.bi.biSize = sizeof(video.bi);
    video.bi.biBitCount = 0;
    HDC hdcDisplay = GetDC(NULL);
    HBITMAP hbm = CreateCompatibleBitmap(hdcDisplay, 1, 1);
    GetDIBits(hdcDisplay, hbm, 0, 1, NULL, (BITMAPINFO
*) &video, DIB_RGB_COLORS);
    GetDIBits(hdcDisplay, hbm, 0, 1, NULL, (BITMAPINFO
*) &video, DIB_RGB_COLORS);
    DeleteObject(hbm);
#endif
}
```

```

ReleaseDC(NULL, hdcDisplay);

// our case is ,framebuffer-32bpp, screen-16bpp
if (video.bi.biCompression == BI_BITFIELDS &&
video.bi.biBitCount == 32)
{
    // check bits
    if (video.fields[0] == 0x00ff0000 &&
        video.fields[1] == 0x0000ff00 &&
        video.fields[2] == 0x000000ff)
    {
        // ok, 32bpp, 8888
        return 32;
    }
}
else
if (video.bi.biCompression == BI_BITFIELDS &&
video.bi.biBitCount == 24)
{
    // check bits
    if (video.fields[0] == 0xff0000 &&
        video.fields[1] == 0x00ff00 &&
        video.fields[2] == 0x0000ff)
    {
        // ok, 24bpp, 888
        return 24;
    }
}
return 0;

#else
return 0;
#endif
}

void InitDevice(dword xRes, dword yRes, HWND hWnd)
{
    dword bit_fields_32[3] = {0x00ff0000,    0x0000ff00,
0x000000ff};
    dword bit_fields_24[3] = {0xff0000,      0x00ff00,
0x0000ff};
    dword bit_fields_16[3] = {0xf800,        0x07e0,
0x001f};
    dword bit_fields_12[3] = {0x0f00,        0x00f0,
0x000f};

```

```

    BITMAPINFOHEADER *l_pBih;

    g_xRes = xRes;
    g_yRes = yRes;
    g_Pitch = ((g_xRes+1)>>1) << 2; // amount of DWORDs
needed to store a scanline * sizeof(DWORD)
    g_hWnd = hWnd;

    // create the buffer ehre so that people can start
drawing before the other thread
    // goes running
    g_pVPage = new word [((g_Pitch * yRes)>>1) + 2]; //
Eyal: i added one for filler. // Yoni: with new pitch it
shouldnt be a problem but added 2 for fun

    // gdi
    l_pBibuf = new byte[sizeof(BITMAPINFOHEADER)+12];
    l_pBi = (BITMAPINFO *)l_pBibuf;
    l_pBih = &l_pBi->bmiHeader;

    // create an internal RGB DIB
    l_pBih->biSize = sizeof(BITMAPINFOHEADER);
    l_pBih->biPlanes = 1;
    l_pBih->biCompression = BI_BITFIELDS;
    l_pBih->biSizeImage = 0;
    l_pBih->biXPelsPerMeter = 0;
    l_pBih->biYPelsPerMeter = 0;
    l_pBih->biClrUsed = 0;
    l_pBih->biClrImportant = 0;

    l_pBih->biWidth = g_xRes;
    l_pBih->biHeight = -(sdword)g_yRes;

    l_pf = IdentifyWindowPF();

    l_pBih->biBitCount = (word)(l_pf ? l_pf : 16);

#ifdef _FLY_OVER_WIN_CE
    // l_pBi fields for conversion
    memcpy(l_pBi->bmiColors, l_pf ? (l_pf == 32 ?
bit_fields_32 : bit_fields_24) : bit_fields_16, 3*4);
    // create convert buffers if needed
    if (l_pf)
    {

```

```

        l_pPage3 = new byte[g_xRes*g_yRes*(l_pf>>3)];
    }

#else
    memcpy(l_pBi->bmiColors, l_pf ? (bit_fields_12) :
bit_fields_16, 3*4);
    if (l_pf)
    {
        ASSERT((g_xRes & 1) == 0);
        ULONG size = g_xRes*g_yRes*l_pf;
        size >>= 3;
        l_pPage3 = new byte[size + 1];
        // l_hBitmap = CreateBitmap( g_xRes, g_yRes, 1,
l_pBih->biBitCount, l_pPage3 );
    }
    else
    {
        // l_hBitmap = CreateBitmap( g_Pitch>>1, g_yRes, 1,
l_pBih->biBitCount, g_pVPage );
        HDC hDC = GetDC(g_hWnd);
        l_hBitmap = CreateDIBSection( hDC, l_pBi,
DIB_RGB_COLORS, &l_pBits, NULL, 0 );
        ReleaseDC( g_hWnd, hDC );
    }
#endif
}

void KillDevice()
{
    if (l_pPage3)
        delete l_pPage3;

    delete l_pBibuf;

    delete g_pVPage;

#ifdef _FLY_OVER_WIN_CE
    if (l_hBitmap) DeleteObject(l_hBitmap);
#endif
}

void ClearWindow()
{
    //memset(g_pVPage, 0x10, g_xRes*g_yRes*2);
    memset(g_pVPage, 0, g_Pitch*g_yRes);
}

```

```
}
```

```
// fps-compensator and d-time calculator
static const dword l_tmr_slp_ratio = 1000 / TMR_RESO;
static dword l_max_fps = 30;
static dword l_ticks_per_frame = TMR_RESO / l_max_fps;
static dword l_glttime = 0; // for gdttime calculation
```

```
void FlipWindow()
```

```
{
    // do we need to convert ?
    if (l_pf)
    {
        dword num;

        num = g_xRes*g_yRes;

        if (l_pf == 32)
        {
#ifdef _FLY_OVER_WIN_CE
            __asm
            {
                push edi
                push esi

                mov ecx,num
                mov edi,l_pPage3
                mov esi,g_pVPage

                lea edi,[edi+ecx*4]
                lea esi,[esi+ecx*2]
                xor ecx,-1
                inc ecx

                fast16232_inner:
                    mov ax,word ptr [esi+ecx*2]
                    shl eax,3
                    ror eax,8
                    shl ax,2
                    shl ah,3
                    rol eax,8

                    mov dword ptr [edi+ecx*4],eax
                    inc ecx
                jnz fast16232_inner
            }
#endif
        }
    }
}
```



```

        pop esi
        pop edi
    }
#else
    ULONG *pDest = (ULONG *)l_pPage3;
    USHORT *pSrc = (USHORT *)g_pVPage;
    ULONG x;
    for (dword i = 0; i < num; i++)
    {
        x = *pSrc++;
        *pDest++ = ((x & 0x1F) << 3) + ((x &
0x7E0) << 5) + ((x & 0xF800) << 8);
    }
#endif
    }
    else if (l_pf == 12)
    {
        UCHAR *pDest = (UCHAR *)l_pPage3;
        USHORT *pSrc = (USHORT *)g_pVPage;
        ULONG x, xx;
        num >>= 1;
        for (dword i = 0; i < num; i++)
        {
            x = *pSrc++;
            xx = *pSrc++;
            *pDest++ = (UCHAR)(((x & 0x1E) >> 1) +
((x & 0x780) >> 3));
            *pDest++ = (UCHAR)(((x & 0xF000) >> 12)
+ ((xx & 0x1E) << 3));
            *pDest++ = (UCHAR)(((xx & 0x780) >> 7)
+ ((xx & 0xF000) >> 8));
        }
    }
    else if (l_pf == 24)
    {
    }
}

// flip
HDC      hDC;
hDC = GetDC(g_hWnd);

```

```

#ifdef _FLY_OVER_WIN_CE

```

```

        StretchDIBits( hDC,
                        0, 0, g_xRes, g_yRes,
                        0, 0, g_xRes, g_yRes,
                        l_pf ? (byte*)l_pPage3 :
(byte*)g_pVPage, l_pBi, DIB_RGB_COLORS, SRCCOPY);
#else
    HDC hDCMemory = CreateCompatibleDC(hDC);

    // HBITMAP hOldBmp2 = (HBITMAP)SelectObject(hDCMemory,
CreateCompatibleBitmap(hDC, g_xRes, g_yRes));

    // memset(g_pVPage, 0xEF, g_xRes * g_yRes * 2);
    // for (int i = 0; i < g_xRes * g_yRes * 3 / 2; i++)
    ((UCHAR*)l_pPage3)[i] = (UCHAR)i;
    // if (l_pf) l_hBitmap = CreateBitmap( g_xRes, g_yRes, 1,
l_pf, l_pPage3 );
    // else l_hBitmap = CreateBitmap( g_Pitch >> 1, g_yRes,
1, 16, g_pVPage );
    // l_hBitmap = CreateDIBSection( hDC, l_pBi,
DIB_RGB_COLORS, &l_pBits, NULL, 0 );
    memcpy( l_pBits, g_pVPage, g_xRes * g_yRes * 2);

    HBITMAP hOldBmp = (HBITMAP)SelectObject(hDCMemory,
l_hBitmap);

    // HBRUSH br = CreateSolidBrush( 0x000FFFFF0 );
    // RECT rec; rec.top = 10; rec.left = 10; rec.bottom =
100; rec.right = 100;
    // FillRect( hDCMemory, &rec, br);

    BitBlt(hDC, 0, 0, g_xRes, g_yRes, hDCMemory, 0, 0,
SRCCOPY);

    SelectObject(hDCMemory, hOldBmp);
    DeleteDC(hDCMemory);

    // DeleteObject( hOldBmp );
    // DeleteObject( l_hBitmap ); l_hBitmap = NULL;
#endif

    ReleaseDC(g_hWnd, hDC);

    // timing

    // calculate global-delta-time (frame time)
    g_gdtime = g_timer - l_gltime;

```

```
g_fgdttime = (float)g_gdtime / (float)TMR_RESO;  
l_gltime = g_timer;  
  
// fps compensator  
if ( g_gdtime < l_ticks_per_frame )  
    Sleep((l_ticks_per_frame-  
g_gdtime)*l_tmr_slp_ratio);  
}
```

Module: Raster.cpp

---

```
// tri-mapper vars
flat_intrp d12,d13,d23;
long Ptr__;

inline void F1st__(long *i,float f)
{
    *i = (long)(f+0.5f); // slow C code
}

// this is actually a slow 16-bit flat inner
static void Flat_Inner16()
{
    // Original ASM inner loop

    if (X2_<=X1_) return;

    //word *p = (word *)Ptr__;
    //p[X1_] = FColor_;
    //p[X2_] = FColor_;
    //return;

#ifdef _FLY_OVER_WIN_CE
    __asm
    {
        Mov EDI,[Ptr__]
        Mov EBX,[X1_]
        LEA EDI,[EDI+EBX*2]
        Mov AX,[FColor_]
        Mov ECX,[X2_]
        Sub ECX,EBX
        Rep StosW
    }
#else
    // slow C code
    long x;
    word *p = (word *)Ptr__ + X1_;
    for(x=X1_;x<X2_;x++)
        *p++ = *p + FColor_;
    //long x;
#endif
}
```

```

static void Flat_Inner16T()
{
    if (X2_<=X1_) return;

#ifdef _FLY_OVER_WIN_CE
    // x86 non-MMX 50% transparent 16bit flat inner
    __asm
    {
        Mov EDI,[Ptr_]
        Mov EBX,[X1_]
        Mov ECX,[X2_]
        Mov AX,[FColor_]
        LEA EDI,[EDI+ECX*2]
        And AX,0F7DEh
        Sub EBX,ECX
        Shr AX,1
        Mov ECX,EBX
        INNER:
            Mov BX,[EDI+ECX*2]
            And BX,0F7DEh
            Shr BX,1
            Add BX,AX
            Mov [EDI+ECX*2],BX
            Inc ECX
            JNZ short INNER
    }
#else
    // slow C code
    long x;
    word addCol = (FColor_ & 0xF7DE) >> 1;
    word *p = (word *)Ptr_ + X1_;
    for(x=X1_;x<X2_;x++)
        *p++ = ((*p & 0xF7DE) >> 1) + addCol;
    //long x;
    //word *p = (word *)Ptr_;
    //p[X1_] = FColor_;
    //p[X2_] = FColor_;
#endif
}

static void (*Flat_Inner)();

void FAST_Flat_Outer(flat_intrp *Left,flat_intrp
*Right,long H_)
{
    if (H_<=0) return;
    do

```

```

{
    Flist__(&X1_, Left->ix-0.499f);
    Flist__(&X2_, Right->ix-0.499f);

    Flat_Inner();

    Left->ix += Left->dx dy;
    Right->ix += Right->dx dy;

    Ptr__ += g_VS.Pitch;
} while (--H_);
}

void FAST_Flat()
{
    long Y1,Y2,Y3,dY12,dY13,dY23;
    float dx12,dy12,subY;
    float Merid;

    Vertex *T;

    //B and C might not be Y-sorted.
    if (A_->PY>B_->PY) {T=A_; A_=B_; B_=T;}
    if (A_->PY>C_->PY) {T=A_; A_=C_; C_=T;}
    if (B_->PY>C_->PY) {T=B_; B_=C_; C_=T;}

    // ok. now, get the polygon drawn.
    // quantize Y values. (ceil(Y))
    Flist__(&Y1,A_->PY+0.499f);
    Flist__(&Y2,B_->PY+0.499f);
    Flist__(&Y3,C_->PY+0.499f);
    dY12 = Y2-Y1; dY13 = Y3-Y1; dY23 = Y3-Y2;

    // calculate d13 deltas
    if (dY13>0)
    {
        subY = (float)Y1-A_->PY;
        d13.dxdy = (C_->PX-A_->PX)/(C_->PY-A_->PY);
        d13.ix = A_->PX + subY*d13.dxdy;

        if (dY12>0)
        {
            dx12 = B_->PX-A_->PX;
            dy12 = B_->PY-A_->PY;
            d12.dxdy = dx12/dy12;
            d12.ix = A_->PX + subY * d12.dxdy;

```

```

        Merid = dx12 - d13.dxdy * dy12;
    } else {Merid = B_>PX-A_>PX;}
    if (dY23>0)
    {
        d23.dxdy = (C_>PX-B_>PX)/(C_>PY-B_>PY);
        subY = (float)Y2-B_>PY;
        d23.ix = B_>PX + subY*d23.dxdy;
    }
} else return;

Ptr__ = ((long)g_VS.FB) + Y1 * g_VS.xRes * 2;

if (Merid>0)
{
    FAST_Flat_Outer(&d13,&d12,dY12);
    FAST_Flat_Outer(&d13,&d23,dY23);
} else {
    FAST_Flat_Outer(&d12,&d13,dY12);
    FAST_Flat_Outer(&d23,&d13,dY23);
}
}

void putpixel(Vertex *V)
{
    //Ptr__ = ((long)g_VS.FB) + Y1 * g_VS.xRes * 2;
    word * w = ((word *)g_VS.FB) + (int)V->PX + ((int)V->PY) * g_VS.xRes;

    word r = *w;
    if ((!r) || r > FColor_) *w = FColor_;
}

void RasterizerSetOP(dword Opcode)
{
    switch (Opcode)
    {
    case 0:
        Flat_Inner = Flat_Inner16;
        break;
    case ROP_Transparent:
        Flat_Inner = Flat_Inner16T;
        break;
    }
}

void RasterizerSetColor(word c)
{

```

```

        FColor_ = c;
    }

void Rasterizer(Vertex **VA)
{
    for(mword i=2; i<C_NumVtx; i++)
    {
        A_ = VA[0];
        B_ = VA[i-1];
        C_ = VA[i];
        //FColor_ = Palette[FillColor_];
        //if (GetAsyncKeyState('W'))
            FAST_Flat();
    }
    //else {
//        putpixel(A_);
//        putpixel(B_);
//        putpixel(C_);
//    }
}

```



Module: Tex.cpp

---

```
#ifndef _FLY_OVER_WIN_CE
```

```
extern "C"
```

```
{
```

```
    // bihi.asm
```

```
    extern void          P_Texture_16(void);
```

```
    extern void          P_Texture_Transparent_16(void);
```

```
    extern dword  CodeStart, CodeSize; //smc
```

```
};
```

```
#endif
```

```
static    Bi_Triangle    BTriangle[3];
```

```
dword     BTriangledD = (dword)BTriangle,  
          DestinationPageD, CurrentTextureD, BPSL;
```

```
static dword OldProtect;
```

```
dword ddu_computed;
```

```
dword g_lvldiff,dufactor;
```

```
float drzdx,duzdx[12],dvzdx[12];
```

```
float drzdy,duzdy[12],dvzdy[12];
```

```
void CalculatedDU(Vertex **VA)
```

```
{
```

```
    // calculate duzdx,dvzdx
```

```
    register Vertex *A = VA[0];
```

```
    register Vertex *B = VA[1];
```

```
    register Vertex *C = VA[2];
```

```
    float dxy = 1.0f / ((C->PY-A->PY) * (B->PX-A->PX) - (B->  
>PY-A->PY) * (C->PX-A->PX));
```

```
    drzdx      =      ((C->PY-A->PY) * (B->RZ-A->RZ) - (B->  
>PY-A->PY) * (C->RZ-A->RZ)) * dxy;
```

```
    dxy *= (TEXTURE_MUL) / (1<<dufactor);
```

```
    duzdx[0] = ((C->PY-A->PY) * (B->UZ-A->UZ) - (B->PY-A->  
>PY) * (C->UZ-A->UZ)) * dxy;
```

```
    dvzdx[0] = ((C->PY-A->PY) * (B->VZ-A->VZ) - (B->PY-A->  
>PY) * (C->VZ-A->VZ)) * dxy;
```

```
    DWORD I;
```

```
    for(I=1;I<12;I++)
```

```
    {
```

```

        duzdx[I] = duzdx[I-1]*2.0f;
        dvzdx[I] = dvzdx[I-1]*2.0f;
    }
    ddu_computed = 0;
}

void EnableFillerSMC()
// Enable writing to the mappers code-segment
{
#ifdef _FLY_OVER_WIN_CE
    VirtualProtect( (void *)CodeStart,
                    CodeSize,
                    PAGE_EXECUTE_READWRITE,
                    &OldProtect);
#endif
}

void DisableFillerSMC()
// simply return to prior VirtualProtect Situation
{
#ifdef _FLY_OVER_WIN_CE
    dword OldOldProtect;

    VirtualProtect( (void *)CodeStart,
                    CodeSize,
                    OldProtect,
                    &OldOldProtect);
#endif
}

static Vertex *VSwap[12];

void Texture_Mapper(Face *F,Vertex **VA)
{
    // before removing eyal+gilad memory trick
    /*   if (0 && GetAsyncKeyState('P'))
    // end - before removing eyal+gilad memory trick
    {
        Vertex *A = VA[0];
        Vertex *B = VA[1];
        Vertex *C = VA[2];

        BTriangle[0].FX=A->PX;
        BTriangle[0].FY=A->PY;
        BTriangle[0].X=(int)A->PX;
        BTriangle[0].Y=(int)A->PY;
        BTriangle[0].Z=A->RZ;
    }
}

```

```

BTriangle[0].U=A->UZ*64.0f;
BTriangle[0].V=A->VZ*64.0f;

BTriangle[1].FX=B->PX;
BTriangle[1].FY=B->PY;
BTriangle[1].X=(int)B->PX;
BTriangle[1].Y=(int)B->PY;
BTriangle[1].Z=B->RZ;
BTriangle[1].U=B->UZ*64.0f;
BTriangle[1].V=B->VZ*64.0f;

BTriangle[2].FX=C->PX;
BTriangle[2].FY=C->PY;
BTriangle[2].X=(int)C->PX;
BTriangle[2].Y=(int)C->PY;
BTriangle[2].Z=C->RZ;
BTriangle[2].U=C->UZ*64.0f;
BTriangle[2].V=C->VZ*64.0f;

CurrentTextureD = (dword)(F->Txtr);
P_Texture_16();

return;
}

long i;
for(i=0;i<C_NumVtx;i++)
    VSwap[i] = VA[C_NumVtx-1-i];

test_texture(F,VSwap,C_NumVtx);
return;

//if
BOOL doAgain = true;
while (doAgain)
{
    int mny = 0, ly = 2, ry = 1;
    if (VA[mny]->PY > VA[1]->PY)
    {
        mny = 1; ly = 0; ry = 2;
    }
    if (VA[mny]->PY > VA[2]->PY)
    {
        mny = 2; ly = 1; ry = 0;
    }
}

```

```

        if ((VA[mny]->PY == VA[ly]->PY) && (VA[mny]->PY ==
VA[ry]->PY)) return;

        float dx1 = VA[ry]->PX - VA[mny]->PX;
        float dy1 = VA[ry]->PY - VA[mny]->PY;

        float dx2 = VA[ly]->PX - VA[mny]->PX;
        float dy2 = VA[ly]->PY - VA[mny]->PY;

        //if ( (dx1 / dy1) > (dx2 / dy2) ) // Bizayon.
        if ( (dx1 * dy2) > (dx2 * dy1) )
        {
            test_texture(F, VA, 3);
        } else
        {
            Vertex *V[3];
            V[0] = VA[2];
            V[1] = VA[1];
            V[2] = VA[0];
            test_texture(F, V, 3);
        }

        doAgain = false;
    }
*/
}

#ifdef _FLY_OVER_WIN_CE
#undef ASSERT
#endif
#define ASSERT(x)

struct VInt
{
    // Vertex *Vtx;
    long X,Y;
};

void test_texture(Face *F,Vertex **VA,dword nVtx)
{
    // return;
    // CW->CCW
    // Vertex *V_;
    // V_ = VA[1]; VA[1]=VA[2]; VA[2] = V_;

```

```

/*  dword      i;
    long upLeftID, upRightID;
    float      lowY = (float)g_yRes;

    long partsAmount = 0;
    _parts *mp = (_parts *)(&txtrstruct.addXL1);

    VInt VI[16];
//  VInt VS[8];

    BOOL zero = false;
    for(i=0;i<nVtx;i++)
    {
//      VI[i].Vtx=VA[i];
      VI[i].X=(long)VA[i]->PX;
      VI[i].Y=(long)VA[i]->PY;

      if (VI[i].X == 0) zero = true;
    }

    // find minimal-Y vertex
    upLeftID = 0;
    for (i = 0; i < nVtx; ++i)
    {
        if (VI[i].Y < VI[upLeftID].Y)
        {
            upLeftID = i;
        }
        else if (VI[i].Y == VI[upLeftID].Y)
        {
            if (VI[i].X < VI[upLeftID].X)
            {
                upLeftID = i;
            }
        }
    }

    upRightID = upLeftID;
    i = upRightID + 1;
    if (i >= nVtx) i = 0;
    while (VI[i].Y == VI[upLeftID].Y)
    {
        upRightID = i++;
        if (i >= nVtx) i = 0;
        if (upRightID == upLeftID) // one scan line
            polygon

```

```

        {
            upLeftID = 0;
            for (long k = 0; k < nVtx; k++)
            {
                if (VI[upLeftID].X > VI[k].X) upLeftID
= k;
            }
            mp->addXL = 0;
            mp->addXR = 0;
            mp->addZ = 0;
            mp->addUZ = 0;
            mp->addVZ = 0;
            mp->D = -1;
            partsAmount = 1;
            break;
        }
    }
    // upest left vertex is in upLeftID, and upRightID
    ...

    long maxY = VI[upRightID].Y;

    i = upRightID + 1;
    if (i >= nVtx) i = 0;

    while (maxY <= VI[i].Y)
    {
        maxY = VI[i++].Y;
        if (i >= nVtx) i = 0;
        if (i == upLeftID) break;
    }

    long leftID = upLeftID, rightID = upRightID,
nextLeftID, nextRightID;
    BOOL leftDone = false, rightDone = false;
    long minY = VI[leftID].Y;
    long dLeftY;
    while (minY != maxY)
    {
        if ( partsAmount > 1 ) mp++;
        else if ( partsAmount == 1 ) mp =
&(txtrstruct.parts[0]);

        nextLeftID = leftID - 1;
        if (nextLeftID < 0) nextLeftID = nVtx - 1;
        if (VI[nextLeftID].Y <= VI[leftID].Y)

```

```

    {
        while (VI[nextLeftID].Y == VI[leftID].Y)
        {
            nextLeftID--;
            if (nextLeftID < 0) nextLeftID = nVtx -
1;
        }
        ASSERT(VI[nextLeftID].Y < VI[leftID].Y);
        //
        //
        //
        //
        //
        {
            partsAmount > 20);
            nextLeftID = leftID;
            leftDone = true;
        }
    }

    nextRightID = rightID + 1;
    if (nextRightID >= nVtx) nextRightID = 0;
    if (VI[nextRightID].Y <= VI[rightID].Y)
    {
        while (VI[nextRightID].Y == VI[rightID].Y)
        {
            nextRightID++;
            if (nextRightID >= nVtx) nextRightID =
0;
        }
        ASSERT(VI[nextRightID].Y < VI[rightID].Y);
    }

    dLeftY = (VI[nextLeftID].Y - VI[leftID].Y);
    mp->addXL = ((VI[nextLeftID].X -
VI[leftID].X)*65536) /dLeftY *2;
    mp->addXR = ((VI[nextRightID].X -
VI[rightID].X)*65536) / (VI[nextRightID].Y - VI[rightID].Y)
*2;
    mp->addXL += g_xRes *65536 *2;
    mp->addXR += g_xRes *65536 *2;

    mp->addZ = (VA[nextLeftID]->RZ - VA[leftID]->RZ)
/dLeftY;
    mp->addUZ = (VA[nextLeftID]->UZ - VA[leftID]-
>UZ)*TEXTURE_MUL /dLeftY;
    mp->addVZ = (VA[nextLeftID]->VZ - VA[leftID]-
>VZ)*TEXTURE_MUL /dLeftY;

    if (VI[nextLeftID].Y > VI[nextRightID].Y)
    {
        mp->D = VI[nextRightID].Y - minY;
    }

```

```

        minY = VI[nextRightID].Y;
        rightID = nextRightID;
    }
    else if (VI[nextLeftID].Y < VI[nextRightID].Y)
    {
        mp->D = VI[nextLeftID].Y - minY;
        minY = VI[nextLeftID].Y;
        leftID = nextLeftID;
    }
    else
    {
        mp->D = VI[nextLeftID].Y - minY;
        minY = VI[nextLeftID].Y;
        leftID = nextLeftID;
        rightID = nextRightID;
    }
    partsAmount++;
    ASSERT(partsAmount > 20);
}

mp->D++;

txtrstruct.moreParts = partsAmount - 1;
txtrstruct.bmp = F->Txtr;
txtrstruct.xRes = g_xRes;
txtrstruct.U1 = (dword) (VA[upLeftID]->U *TEXTURE_MUL);
txtrstruct.V1 = (dword) (VA[upLeftID]->V *TEXTURE_MUL);
txtrstruct.DeltaScr1 = (dword) (g_pVPage +
VI[upLeftID].X + (VI[upLeftID].Y * g_xRes));
txtrstruct.DeltaScr2 = (dword) (g_pVPage +
VI[upRightID].X + (VI[upLeftID].Y * g_xRes));
txtrstruct.Z1 = VA[upLeftID]->RZ;
txtrstruct.UZ1 = VA[upLeftID]->UZ *TEXTURE_MUL;
txtrstruct.VZ1 = VA[upLeftID]->VZ *TEXTURE_MUL;

// PATCH:
// txtrstruct.U1 = txtrstruct.UZ1 / txtrstruct.Z1;
// txtrstruct.V1 = txtrstruct.VZ1 / txtrstruct.Z1;
// PATCH:

txtrstruct.adZ = drzdx;
txtrstruct.adUZ = duzdx[g_lvldiff];
txtrstruct.adVZ = dvzdx[g_lvldiff];
txtrstruct.adZdy = drzdy;
txtrstruct.adUZdy = duzdy[g_lvldiff];
txtrstruct.adVZdy = dvzdy[g_lvldiff];

```



```

//  txtrstruct.adZdy = 0;
//  txtrstruct.adUZdy = txtrstruct.addUZ1;
//  txtrstruct.adVZdy = txtrstruct.addVZ1;

{
    Vertex *A = VA[0];
    Vertex *B = VA[1];
    Vertex *C = VA[2];

    double divDxy = ((C->PX-A->PX) * (B->PY-A->PY)) -
((B->PX-A->PX) * (C->PY-A->PY));
    double dxy = 1.0 / divDxy;
    //txtrstruct.adZdy = ((C->PX-A->PX) * (B->RZ-
A->RZ) - (B->PX-A->PX) * (C->RZ-A->RZ)) * dxy;
    dxy *= (TEXTURE_MUL);
    txtrstruct.adUZdy = ((C->PX-A->PX) * (B->UZ-A->UZ) -
(B->PX-A->PX) * (C->UZ-A->UZ)) * dxy;
    txtrstruct.adVZdy = ((C->PX-A->PX) * (B->VZ-A->VZ) -
(B->PX-A->PX) * (C->VZ-A->VZ)) * dxy;
    //  float dxy = 1.0 / ((VI[2].Y-VI[0].Y) * (VI[1].X-
VI[0].X) - (VI[1].Y-VI[0].Y) * (VI[2].X-VI[0].X));
    //  txtrstruct.adZ = ((VI[2].Y-VI[0].Y) * (B->RZ-
A->RZ) - (VI[1].Y-VI[0].Y) * (C->RZ-A->RZ)) * dxy;
    //  dxy *= (TEXTURE_MUL);
    //  txtrstruct.adUZ = ((VI[2].Y-VI[0].Y) * (B->UZ-A-
>UZ) - (VI[1].Y-VI[0].Y) * (C->UZ-A->UZ)) * dxy;
    //  txtrstruct.adVZ = ((VI[2].Y-VI[0].Y) * (B->VZ-A-
>VZ) - (VI[1].Y-VI[0].Y) * (C->VZ-A->VZ)) * dxy;
}

//  txtrstruct.UZ1 += (VA[upLeftID]->PY -
((float)VI[upLeftID].Y)) *txtrstruct.adUZdy;
//  txtrstruct.VZ1 += (VA[upLeftID]->PY -
((float)VI[upLeftID].Y)) *txtrstruct.adVZdy;

    txtrstruct.adZ16 = txtrstruct.adZ * 16;
    txtrstruct.adUZ16 = txtrstruct.adUZ * 16;
    txtrstruct.adVZ16 = txtrstruct.adVZ * 16;

    if (upLeftID != upRightID)
    {
        upLeftID = upLeftID;
    }

//  txtrstruct.U1 = txtrstruct.DeltaScr1;
//  txtrstruct.V1 = txtrstruct.DeltaScr1;

```

```

txtrstruct.bmp = (word *)F->Txtr;

// before removing eyal+gilad memory trick

        // word *wptr = (word *)F->Txtr;
        // txtrstruct.bmp = wptr;+64*64*4;

// end - before removing eyal+gilad memory trick

// For debugging:
if (zero = 0)
{
    static short tBmp[64*64+1];
    memset(tBmp, (BYTE)(((long)txtrstruct.bmp) >> 8)
| 13, (64*64+1)*2);
    txtrstruct.bmp = &tBmp;
}

texture_per64();
}*/

/* Vertex      *pVerts[8];
dword rVtx = nVtx-lowID;
memcpy(VS,VI+lowID,rVtx*sizeof(VInt));
memcpy(VS+rVtx,VI,lowID*sizeof(VInt));

// void *bmp;           // bitmap, should be aligned to
0x20000 0 *
// long DeltaScr1, // addr of first pixel to draw
4 *
// DeltaScr2,           // addr of last pixel in
first line 8 *
// addXL1,             // add to left offset
12
// addXR1,             // add to right offset
16
// D1;                 // size of first part of
the poly 20
// float
// addZ1,              // add to left (1/Z)
24
// addUZ1,             // add to left (U/Z)
28
// addVZ1,             // add to left (V/Z)
32
// long

```

```

//          U1, V1;                      // first point cords
36
//      float
//          adZ16,                      // add after every 16
pixels to (1/Z)      40      *
//          adUZ16,                    // add after every 16 pixels to
(U/Z)      44      *
//          adVZ16,                    // add after every 16 pixels to
(V/Z)      48      *
//          adZ,                      // add after every pixel
to (1/Z)      52      *
//          adUZ,                    // add after every pixel to
(U/Z)      56      *
//          adVZ,                    // add after every pixel to
(V/Z)      60      *
//          Z1,                      // first (1/Z)
64
//          UZ1,                    // first (U/Z)
68
//          VZ1;                    // first (V/Z)
72
//      long moreParts;      // how much parts to the polygon(
-1)      76
//
//      struct _parts
//      {
//          addXL,      // add to left offset
80+24K *
//          addXR,      // add to right offset
*
//          D,          // size of part
*
//          addZ,      // add to left (1/Z)
*
//          addUZ,      // add to left (U/Z)
*
//          addVZ,      // add to left (V/Z)
*
//      } parts[20];

long partsAmount = 0;
_parts *mp = (_parts *)(&txtrstruct.addXL1);

// verify that the total height is nonzero
//if (p[0][1] == p[2][1]) return;

long leftID = lowID, rightID = lowID;

```

```

    long nextLeftID = leftID, nextRightID = rightID;

    if (++nextLeftID >= nVtx) nextLeftID = 0;
    if (--nextRightID < 0) nextRightID = nVtx-1;

    txtrstruct.DeltaScr1 = (dword)(g_pVPage + VS[lowID].X
+ (VS[lowID].Y * g_xRes));

    if (VS[lowID].Y == VS[rightID].Y)
        txtrstruct.DeltaScr2 = (dword)(g_pVPage +
VS[leftID].X + (VS[leftID].Y * g_xRes));
    else
        txtrstruct.DeltaScr2 = txtrstruct.DeltaScr1;

    txtrstruct.bmp = F->Txtr;
    txtrstruct.U1 = (dword)(A->U * 64*256);
    txtrstruct.V1 = (dword)(A->V * 64*256);

    bool go = true;
    while (go)
    {
        bool addPart;
        addPart = false;
        txtrstruct.addXL1 = ((p[2][0] - p[0][0]) * 65536)
/ (p[2][1] - p[0][1]) * 2;
        txtrstruct.addXL2 = ((p[2][0] - p[1][0]) * 65536)
/ (p[2][1] - p[1][1]) * 2;
        txtrstruct.D1 = p[2][1] - p[0][1];

        txtrstruct.addZ1 = (C->RZ - A->RZ) / (p[2][1] -
p[0][1]);
        txtrstruct.addUZ1 = ((C->UZ - A->UZ) * 64*256) /
(p[2][1] - p[0][1]);
        txtrstruct.addVZ1 = ((C->VZ - A->VZ) * 64*256) /
(p[2][1] - p[0][1]);

    }

    if (p[0][1] == p[1][1])
    {
        txtrstruct.DeltaScr2 = (dword)(g_pVPage + B->PX
+ (B->PY * g_xRes));
        txtrstruct.addXL1 = ((p[2][0] - p[0][0]) * 65536)
/ (p[2][1] - p[0][1]) * 2;
        txtrstruct.addXL2 = ((p[2][0] - p[1][0]) * 65536)
/ (p[2][1] - p[1][1]) * 2;
    }

```

```

        txtrstruct.D1 = p[2][1] - p[0][1];

        txtrstruct.addZ1 = (C->RZ - A->RZ) / (p[2][1] -
p[0][1]);
        txtrstruct.addUZ1 = ((C->UZ - A->UZ) *64*256) /
(p[2][1] - p[0][1]);
        txtrstruct.addVZ1 = ((C->VZ - A->VZ) *64*256) /
(p[2][1] - p[0][1]);
        if (p[1][0] - p[0][0] != 0)
        {
            txtrstruct.adZ = (C->RZ - A->RZ) / (p[1][0]
- p[0][0]);
            txtrstruct.adUZ = ((C->UZ - A->UZ) *64*256)
/ (p[1][0] - p[0][0]);
            txtrstruct.adVZ = ((C->VZ - A->VZ) *64*256)
/ (p[1][0] - p[0][0]);
            txtrstruct.adZ16 = txtrstruct.adZ * 16;
            txtrstruct.adUZ16 = txtrstruct.adUZ16 * 16;
            txtrstruct.adVZ16 = txtrstruct.adVZ16 * 16;
        }
        else
        {
            txtrstruct.adZ = 0;
            txtrstruct.adUZ = 0;
            txtrstruct.adVZ = 0;
            txtrstruct.adZ16 = 0;
            txtrstruct.adUZ16 = 0;
            txtrstruct.adVZ16 = 0;
        }
    }
    else
    {
        txtrstruct.DeltaScr2 = txtrstruct.DeltaScr1;
        if (p[1][1] == p[2][1])
        {
            txtrstruct.addXL1 = ((p[2][0] - p[0][0])
*65536) / (p[2][1] - p[0][1]) *2;
            txtrstruct.addXL2 = ((p[2][0] - p[1][0])
*65536) / (p[2][1] - p[1][1]) *2;
            txtrstruct.D1 = p[2][1] - p[0][1];

            txtrstruct.addZ1 = (C->RZ - A->RZ) /
(p[2][1] - p[0][1]);
            txtrstruct.addUZ1 = ((C->UZ - A->UZ)
*64*256) / (p[2][1] - p[0][1]);
            txtrstruct.addVZ1 = ((C->VZ - A->VZ)
*64*256) / (p[2][1] - p[0][1]);

```

```

        }
        else
        {
            }
    }

    texture_per64();*/
}

// Perspective-corrected Subpixeling/Subtexeling 16bit
Texture mapper
// Advanced version scanconverts ngons directly (operative)
// Simple version Triangulates input Polygons. (operative)
// Next: Interfacing version usable with Eyal's
Texturemapper
static Vertex *RR_VA_[25],**RR_VA=RR_VA_+1;
static long RR_Y_[25],*RR_Y = RR_Y_+1;
static dword RR_NumVerts;
static word *RR_TexturePtr;
static dword RR_SectionHeight;
static dword RR_GPositive;
static float LeftX,RightX;
static float LeftUZ,LeftVZ,LeftRZ;
static float LeftDX,RightDX;
static float LeftDUZ,LeftDVZ,LeftDRZ;
static float IntUZ,IntVZ,IntRZ;
static float drzdx_,duzdx_,dvzdx_;
static float drzdx_16,duzdx_16,dvzdx_16;
static float Inv16[15] = {
    1.0f/1.0f, 1.0f/2.0f, 1.0f/3.0f, 1.0f/4.0f, 1.0f/5.0f,
    1.0f/6.0f, 1.0f/7.0f, 1.0f/8.0f, 1.0f/9.0f,
    1.0f/10.0f,
    1.0f/11.0f, 1.0f/12.0f, 1.0f/13.0f, 1.0f/14.0f,
    1.0f/15.0f
};
static word *Scanline,*ScanPtr;
static long Width;

static Texture RR_Texture;
static mword RR_MaskU, RR_MaskV;

#define RR_TScale (65536.0f)
float RR_TScaleU, RR_TScaleV;

void RR_SetTexture(Texture *Tx)

```

```

{
    if (Tx)
    {
        memcpy(&RR_Texture, Tx, sizeof(Texture));
        RR_TexturePtr = (word *)Tx->_data;
        RR_MaskU = ((1<<Tx->_l2xRes)-1)<<16;
        RR_MaskV = ((1<<Tx->_l2yRes)-1)<<16;
        RR_TScaleU = 65536.0*(1<<Tx->_l2xRes);
        RR_TScaleV = 65536.0*(1<<Tx->_l2yRes);
    } else RR_TexturePtr = NULL; // disables the mapper
}

// Automatic inner loop setting
/*void RR_SetTexture(Texture *Tx)
{
    RR_SetTextureParams(Tx);

    if (RR_Texture._stateFlags & TSF_
}*/

// 28.7.01: making an attempt to allow 'plugin' innerloops.
// first generalize usage of Eyal's and the normal mapper
void (*RR_OuterInit)();
void (*RR_OuterLoop)();
void (*RR_InnerLoop)();

dword RGrads(Vertex *A,Vertex *B,Vertex *C)
{
    float dxy = 1.0f / (((C->PY-A->PY)*(B->PX-A->PX))-((B->PY-A->PY)*(C->PX-A->PX)));
    drzdx_ = ((C->PY-A->PY)*(B->RZ-A->RZ)-(B->PY-A->PY)*(C->RZ-A->RZ)) * dxy;
    drzdx_16 = drzdx_ * 16.0f;
    //dxy *= RR_TScale;

    duzdx_ = ((C->PY-A->PY)*(B->UZ-A->UZ)-(B->PY-A->PY)*(C->UZ-A->UZ)) * dxy * RR_TScaleU;
    duzdx_16 = duzdx_ * 16.0f;

    dvzdx_ = ((C->PY-A->PY)*(B->VZ-A->VZ)-(B->PY-A->PY)*(C->VZ-A->VZ)) * dxy * RR_TScaleV;
    dvzdx_16 = dvzdx_ * 16.0f;

    return 0;
}

```

```
}
```

```
// An advanced version of RR_InnerLoop, coded in x86  
assembly. This function should be optimized// for Pentium  
Pro/2/3 Processors.
```

```
// OPT -> Dualpixel processing aligned to 4 bytes
```

```
// OPT -> efficient CPU/FPU sync
```

```
static void RR_InnerLoop_Ax86P2opt()
```

```
{
```

```
    static const float one = 1.0f;
```

```
    static float rrz;
```

```
    static long u0,v0,u1,v1;
```

```
/*    __asm {
```

```
        ; Preperations for Inner
```

```
        ; FPU state ST(0..7)
```

```
        fld [IntRZ]          ; RZ
```

```
        fld one              ; RZ, 1
```

```
        fdiv ST,ST(1)        ; RZ, Z
```

```
        fld drzdx_16         ; RZ, Z, dRZ
```

```
        faddp ST(2),ST       ; Z, RZ+dRZ
```

```
        fxch ST,ST(1)        ; RZ+dRZ, Z
```

```
        fstp IntRZ           ; Z
```

```
        fld [dvzdx_16]       ; Z, dVZ
```

```
        fld [IntUZ]          ; Z, dVZ, UZ
```

```
        fld [IntVZ]          ; Z, dVZ, UZ, VZ
```

```
        fld [duzdx_16]       ; Z, dVZ, UZ, VZ, dUZ
```

```
        fxch ST,ST(4)        ; dUZ, dVZ, UZ, VZ, Z
```

```
        fadd ST,ST(2)        ; dUZ+UZ, dVZ, UZ, VZ, Z
```

```
        fxch ST,ST(1)        ; dVZ, dUZ+UZ, UZ, VZ, Z
```

```
        fadd ST,ST(3)        ; dVZ+VZ, dUZ+UZ, UZ, VZ, Z
```

```
        fstp IntVZ           ; dUZ+UZ, UZ, VZ, Z
```

```
        fstp IntUZ           ; UZ, VZ, Z
```

```
        fxch ST,ST(2)        ; Z, VZ, UZ
```

```
        fmul ST(1),ST        ; Z, V, UZ
```

```
        fmulp ST(2),ST       ; V, U
```

```
        fistp [v0]           ; U
```

```
        fistp [u0]           ; /
```

```
}
```

```
    __asm {
```

```
        emms ; EMPTY(RESET) MMX REGISTER STATE
```

```
    }*/
```

```
}
```



```

static void RR_InnerLoop_ZCK()
{
    // Affine-16 Inner: preparation
    static float rrz;
    static long u0,v0,u1,v1;
    rrz = 1.0f / IntrZ;
    u0 = (long)(IntUZ * rrz); // & 0x3FFFFFF;
    v0 = (long)(IntVZ * rrz); // & 0x3FFFFFF;
    IntrZ += drzdx_16;
    IntUZ += duzdx_16;
    IntVZ += dvzdx_16;

    rrz = 1.0f / IntrZ;
    u1 = (long)(IntUZ * rrz); // & 0x3FFFFFF;
    v1 = (long)(IntVZ * rrz); // & 0x3FFFFFF;
    IntrZ += drzdx_16;
    IntUZ += duzdx_16;
    IntVZ += dvzdx_16;

    static long du,dv,W16;
    static long Counter;

    if (Width>=16)
    {
        W16 = Width>>4;

        // affine-16 inner outer
        do {
            rrz = 1.0f / IntrZ;
            du = (u1-u0)>>4;
            dv = (v1-v0)>>4;
            // affine-16 inner inner
            Counter = 16;
            do {
                register word w =
RR_TexturePtr[((u0&0xFF0000)>>16)+((v0&0xFF0000)>>(16-
RR_Texture._12xRes))];
                if (w) *ScanPtr++ = w; else ScanPtr++;
                u0 += du;
                v0 += dv;
            } while (--Counter);
            u1 = (long)(IntUZ * rrz);
            v1 = (long)(IntVZ * rrz);
            IntrZ += drzdx_16;
            IntUZ += duzdx_16;
            IntVZ += dvzdx_16;
        } while (--W16);
    }
}

```

```

        Width &= 0xF;
        if (!Width) return;
    }

    du = (u1-u0)>>4;
    dv = (v1-v0)>>4;
    do {
        /*ScanPtr++ =
RR_TexturePtr[((u0&0xFF0000)>>16)+((v0&0xFF0000)>>(16-
RR_Texture._12xRes))];
        /*ScanPtr++ += 0x7bef;
        register word w =
RR_TexturePtr[((u0&0xFF0000)>>16)+((v0&0xFF0000)>>(16-
RR_Texture._12xRes))];
        if (w) *ScanPtr++ = w; else ScanPtr++;
        u0 += du;
        v0 += dv;
    } while (--Width);
}

// This uses Affine-16 bilinear interpolation,
// also it interpolates last segment out of polygon bounds,
this has to be
// fixed. (ask Rage/IMR)
static void RR_InnerLoop_Standard()
{
    // slow inner
    /*do {
        float rrz = 1.0f / IntrZ;
        long u = ((long) (IntUZ * rrz)>>16) & 0x3F;
        long v = ((long) (IntVZ * rrz)>>16) & 0x3F;
        *ScanPtr++ = RR_TexturePtr[(u+(v<<6))];
        IntrZ += drzdx_;
        IntUZ += duzdx_;
        IntVZ += dvzdx_;
    } while (--Width);
    return;*/

    // Affine-16 Inner: preparation
    static float rrz;
    static long u0,v0,u1,v1;
    rrz = 1.0f / IntrZ;
    u0 = (long) (IntUZ * rrz);// & 0x3FFFFFF;
    v0 = (long) (IntVZ * rrz);// & 0x3FFFFFF;
    IntrZ += drzdx_16;
    IntUZ += duzdx_16;
    IntVZ += dvzdx_16;

```

```

rrz = 1.0f / IntrZ;
u1 = (long)(IntUZ * rrz); // & 0x3FFFFFF;
v1 = (long)(IntVZ * rrz); // & 0x3FFFFFF;
IntrZ += drzdx_16;
IntUZ += duzdx_16;
IntVZ += dvzdx_16;

static long du,dv,W16;
static long Counter;

if (Width>=16)
{
    W16 = Width>>4;

    // affine-16 inner outer
    do {
        rrz = 1.0f / IntrZ;
        du = (u1-u0)>>4;
        dv = (v1-v0)>>4;
        // affine-16 inner inner
        Counter = 16;
        do {
            *ScanPtr++ =
RR_TexturePtr[((u0&RR_MaskU)>>16)+((v0&RR_MaskV)>>(16-
RR_Texture._12xRes))];
            /*ScanPtr++ += 0x7bef;
            u0 += du;
            v0 += dv;
        } while (--Counter);
        u1 = (long)(IntUZ * rrz);
        v1 = (long)(IntVZ * rrz);
        IntrZ += drzdx_16;
        IntUZ += duzdx_16;
        IntVZ += dvzdx_16;
    } while (--W16);
    Width &= 0xF;
    if (!Width) return;
}

du = (u1-u0)>>4;
dv = (v1-v0)>>4;
do {
    *ScanPtr++ =
RR_TexturePtr[((u0&RR_MaskU)>>16)+((v0&RR_MaskV)>>(16-
RR_Texture._12xRes))];
    /*ScanPtr++ += 0x7bef;

```

```

        u0 += du;
        v0 += dv;
    } while (--Width);
}

static void RR_OuterLoop_Standard()
{
    static float subX;
    if (!RR_SectionHeight) return;
    do {
        long x0 = (long)ceil(LeftX);
        long x1 = (long)ceil(RightX);
        Width = x1-x0;
        if (Width>0) {
            ScanPtr = Scanline + x0;
            subX = x0 - LeftX;
            IntUZ = LeftUZ + subX * duzdx_;
            IntVZ = LeftVZ + subX * dvzdx_;
            IntRZ = LeftRZ + subX * drzdx_;
            RR_InnerLoop();
        }
        LeftX += LeftDX;
        LeftUZ += LeftDUZ;
        LeftVZ += LeftDVZ;
        LeftRZ += LeftDRZ;
        RightX += RightDX;
        Scanline += g_VS.xRes;
    } while (--RR_SectionHeight);
}

```

```

// Interface functions with eyal's mapper
static void RR_OuterInit_Standard()
{
    RRGrads(RR_VA[0],RR_VA[1],RR_VA[2]);
}

```

```

#ifdef _FLY_OVER_WIN_CE
static void RR_OuterInit_Interface()
{
    // why the d [1/z,u/z,v/z] / d [x,y] arent identical
    across the entire plane..
    // didnt figure...probably has to do with the linear
    discontinuities of the mapping
    // coordinates between polygons
}

```

```

        // Per-polygon delta computation vars
#ifdef HYPERBOLIC_MAPPING
        double divDxy,dxy;
        Vertex *A = RR_VA[0];
        Vertex *B = RR_VA[1];
        Vertex *C = RR_VA[2];
#endif

        txtrstruct.bmp = RR_TexturePtr;
        txtrstruct.scanLinePtr = Scanline;
        txtrstruct.Pitch = g_VS.Pitch;

#ifdef HYPERBOLIC_MAPPING
        txtrstruct.adZ = drzdx;
        txtrstruct.adUZ = duzdx[g_lvldiff];// * RR_TScale;
        txtrstruct.adVZ = dvzdx[g_lvldiff];// * RR_TScale;
#else
        // Per-polygon d/dX computation (required if roll!=0)
        divDxy = ((C->PY-A->PY) * (B->PX-A->PX)) - ((B->PY-A-
>PY) * (C->PX-A->PX));
        dxy = 1.0 / divDxy;
        txtrstruct.adZ = ((C->PY-A->PY) * (B->RZ-A->RZ) - (B->PY-
A->PY) * (C->RZ-A->RZ)) * dxy;
        dxy *= (TEXTURE_MUL);
        txtrstruct.adUZ = ((C->PY-A->PY) * (B->UZ-A->UZ) - (B->PY-
A->PY) * (C->UZ-A->UZ)) * dxy;
        txtrstruct.adVZ = ((C->PY-A->PY) * (B->VZ-A->VZ) - (B->PY-
A->PY) * (C->VZ-A->VZ)) * dxy;
#endif

#ifdef HYPERBOLIC_MAPPING
        txtrstruct.adZdy = drzdy;
        txtrstruct.adUZdy = duzdy[g_lvldiff];// * RR_TScale;
        txtrstruct.adVZdy = dvzdy[g_lvldiff];// * RR_TScale;
#else
        // Per-polygon d/dY computation (always required)
        divDxy = ((C->PX-A->PX) * (B->PY-A->PY)) - ((B->PX-A-
>PX) * (C->PY-A->PY));
        dxy = 1.0 / divDxy;
        txtrstruct.adZdy = ((C->PX-A->PX) * (B->RZ-A->RZ) - (B-
>PX-A->PX) * (C->RZ-A->RZ)) * dxy;
        dxy *= (TEXTURE_MUL);
        txtrstruct.adUZdy = ((C->PX-A->PX) * (B->UZ-A->UZ) - (B-
>PX-A->PX) * (C->UZ-A->UZ)) * dxy;
        txtrstruct.adVZdy = ((C->PX-A->PX) * (B->VZ-A->VZ) - (B-
>PX-A->PX) * (C->VZ-A->VZ)) * dxy;
#endif

```

```

}

static void RR_OuterLoop_Interface()
{
    /*
    void *bmp; // Texture Ptr
    word *scanLinePtr; // Pointer to start of first
scanline
    long Pitch; // Pitch (in bytes)
    long leftX,rightX;
    long addXL,addXR;
    long rZ,uZ,vZ;
    dword sectionDelta;

    float u1,v1, // [u1,v1] is the texture coordinate on
the left edge in first scanline
        adZ,adUZ,adVZ, // d[1/Z, U/Z, V/Z] / dx
        adZdy,adUZdy,adVZdy;// d[1/Z, U/Z, V/Z] / dx
    */
    txtrstruct.leftX = (long) (LeftX * 65536.0);
    txtrstruct.rightX = (long) (RightX * 65536.0);
    txtrstruct.addXL = (long) (LeftDX * 65536.0);
    txtrstruct.addXR = (long) (RightDX * 65536.0);
    float subX = (float)ceil(LeftX) - LeftX;
    txtrstruct.uZ = LeftUZ + subX * txtrstruct.adUZ;
    txtrstruct.vZ = LeftVZ + subX * txtrstruct.adVZ;
    txtrstruct.rZ = LeftRZ + subX * txtrstruct.adZ;

    float z = 1.0f / txtrstruct.rZ;
    txtrstruct.u1 = txtrstruct.uZ * z;
    txtrstruct.v1 = txtrstruct.vZ * z;
    txtrstruct.sectionDelta = RR_SectionHeight;

    texture_per64();
}
#else // _FLY_OVER_WIN_CE
static void RR_OuterInit_Interface()
{
}
static void RR_OuterLoop_Interface()
{
}
#endif // _FLY_OVER_WIN_CE

dword RR_Gradients(Vertex *A,Vertex *B,Vertex *C)
{

```

```

        float divDxy = ((C->PY-A->PY)*(B->PX-A->PX)) - ((B->PY-
A->PY)*(C->PX-A->PX));
        if (divDxy>0.0000001) {
            RR_GPositive = 1;
        } else if (divDxy<-0.0000001) {
            RR_GPositive = 0;
        } else return 1; //Degenerate triangle (colinear
points)
        float dxy = 1.0f / divDxy;
        drzdx_ = ((C->PY-A->PY)*(B->RZ-A->RZ) - (B->PY-A-
>PY)*(C->RZ-A->RZ)) * dxy;
        drzdx_16 = drzdx_ * 16.0f;
        //dxy *= RR_TScale;

        duzdx_ = ((C->PY-A->PY)*(B->UZ-A->UZ) - (B->PY-A-
>PY)*(C->UZ-A->UZ)) * dxy * RR_TScaleU;
        duzdx_16 = duzdx_ * 16.0f;

        dvzdx_ = ((C->PY-A->PY)*(B->VZ-A->VZ) - (B->PY-A-
>PY)*(C->VZ-A->VZ)) * dxy * RR_TScaleV;
        dvzdx_16 = dvzdx_ * 16.0f;

        return 0;
}

```

```

void Referance_Rasterizer_Triangle()
{
    // Sort input Vertices by Y coordinate.
    Vertex *A = RR_VA[0], *B = RR_VA[1], *C = RR_VA[2], *T;
    if (A->PY > B->PY) {T=A; A=B; B=T;}
    if (A->PY > C->PY) {T=A; A=C; C=T;}
    if (B->PY > C->PY) {T=B; B=C; C=T;}

    if (RR_Gradients(A,B,C)) return;

    // round down Y coordinates
    long y0 = (long)ceil(A->PY);
    long y1 = (long)ceil(B->PY);
    long y2 = (long)ceil(C->PY);

    float subY; //sub-pixel correction values
    float idy; // inverse dy

    Scanline = (word *)g_VS.FB + y0 * (g_VS.Pitch>>1);
}

```

```

// Check for TOP edge
if (y0==y1)
{
    // Check for flat polygon
    if (y0==y2) return;

    if (RR_GPositive)
    {
        // Calculate 0-2, 1-2 edge delta values
        RightDX = (C->PX-B->PX)/(C->PY-B->PY);
        idy = 1.0f / (C->PY-A->PY);
        LeftDX = (C->PX-A->PX) * idy;
        LeftDRZ = (C->RZ-A->RZ) * idy;
        //idy *= RR_TScale;
        LeftDUZ = (C->UZ-A->UZ) * idy * RR_TScaleU;
        LeftDVZ = (C->VZ-A->VZ) * idy * RR_TScaleV;

        // section is 0->2|1->2, apply subpixelling
correction
        subY = y0 - A->PY;
        LeftX = A->PX + subY * LeftDX;
        LeftUZ = A->UZ * RR_TScaleU + LeftDUZ *
subY;
        LeftVZ = A->VZ * RR_TScaleV + LeftDVZ *
subY;
        LeftRZ = A->RZ + LeftDRZ * subY;
        RightX = B->PX + (y1 - B->PY) * RightDX;
    } else {
        // Calculate 1-2, 0-2 edge delta values
        RightDX = (C->PX-A->PX)/(C->PY-A->PY);
        idy = 1.0f / (C->PY-B->PY);
        LeftDX = (C->PX-B->PX) * idy;
        LeftDRZ = (C->RZ-B->RZ) * idy;
        //idy *= RR_TScale;
        LeftDUZ = (C->UZ-B->UZ) * idy * RR_TScaleU;
        LeftDVZ = (C->VZ-B->VZ) * idy * RR_TScaleV;

        // section is 1->2|0->2, apply subpixelling
correction
        subY = y1 - B->PY;
        LeftX = B->PX + subY * LeftDX;
        LeftUZ = B->UZ * RR_TScaleU + LeftDUZ *
subY;
        LeftVZ = B->VZ * RR_TScaleV + LeftDVZ *
subY;
        LeftRZ = B->RZ + LeftDRZ * subY;
        RightX = A->PX + (y0 - A->PY) * RightDX;
    }
}

```



```

    }

    // Jump to position after all preparations for 1-
2 section
    goto Section12;
}

// Section 0-1:
if (RR_GPositive)
{
    // Calculate 0-2, 0-1 edge delta values
    RightDX = (B->PX-A->PX)/(B->PY-A->PY);
    idy = 1.0f / (C->PY-A->PY);
    LeftDX = (C->PX-A->PX) * idy;
    LeftDRZ = (C->RZ-A->RZ) * idy;
    //idy *= RR_TScale;
    LeftDUZ = (C->UZ-A->UZ) * idy * RR_TScaleU;
    LeftDVZ = (C->VZ-A->VZ) * idy * RR_TScaleV;
} else {
    // Calculate 0-1, 0-2 edge delta values
    RightDX = (C->PX-A->PX)/(C->PY-A->PY);
    idy = 1.0f / (B->PY-A->PY);
    LeftDX = (B->PX-A->PX) * idy;
    LeftDRZ = (B->RZ-A->RZ) * idy;
    //idy *= RR_TScale;
    LeftDUZ = (B->UZ-A->UZ) * idy * RR_TScaleU;
    LeftDVZ = (B->VZ-A->VZ) * idy * RR_TScaleV;
}
// apply subpixelling correction
subY = y0 - A->PY;
LeftX = A->PX + subY * LeftDX;
LeftUZ = A->UZ * RR_TScaleU + LeftDUZ * subY;
LeftVZ = A->VZ * RR_TScaleV + LeftDVZ * subY;
LeftRZ = A->RZ + LeftDRZ * subY;
RightX = A->PX + subY * RightDX;
RR_SectionHeight = y1-y0;
RR_OuterLoop();

subY = y1 - B->PY;

if (RR_GPositive)
{
    // Calculate 1-2 edge delta values
    RightDX = (C->PX-B->PX)/(C->PY-B->PY);
    // apply subpixelling correction
    RightX = B->PX + subY * RightDX;
} else {

```

```

        // Calculate 1-2 edge delta values
        idy = 1.0f / (C->PY - B->PY);
        LeftDX = (C->PX-B->PX) * idy;
        LeftDRZ = (C->RZ-B->RZ) * idy;
        //idy *= RR_TScale;
        LeftDUZ = (C->UZ-B->UZ) * idy * RR_TScaleU;
        LeftDVZ = (C->VZ-B->VZ) * idy * RR_TScaleV;

        // apply subpixelling correction
        LeftX = B->PX + subY * LeftDX;
        LeftUZ = B->UZ * RR_TScaleU + LeftDUZ * subY;
        LeftVZ = B->VZ * RR_TScaleV + LeftDVZ * subY;
        LeftRZ = B->RZ + LeftDRZ * subY;
    }

Section12::
    RR_SectionHeight = y2-y1;
    RR_OuterLoop();
}

static dword Left,Right,HLeft,HRight;

void RR_GetLeftSection()
{
    long cy = RR_Y[Left],ny;
    do
    {
        if (Left==RR_NumVerts-1) Left=0;
        else ++Left;
        ny = RR_Y[Left];
    } while (ny==cy);
    if (ny<cy) return; // End of polygon
    HLeft = ny-cy;
    float subY = cy - RR_VA[Left-1]->PY;
    float idy = 1.0f / (RR_VA[Left]->PY - RR_VA[Left-1]-
>PY);
    LeftDX = (RR_VA[Left]->PX - RR_VA[Left-1]->PX) * idy;
    LeftDRZ = (RR_VA[Left]->RZ - RR_VA[Left-1]->RZ) * idy;
    //idy *= RR_TScale;
    LeftDUZ = (RR_VA[Left]->UZ - RR_VA[Left-1]->UZ) * idy
* RR_TScaleU;
    LeftDVZ = (RR_VA[Left]->VZ - RR_VA[Left-1]->VZ) * idy
* RR_TScaleV;

    LeftX = RR_VA[Left-1]->PX + LeftDX * subY;
    LeftUZ = RR_VA[Left-1]->UZ * RR_TScaleU + LeftDUZ *
subY;

```

```

    LeftVZ = RR_VA[Left-1]->VZ * RR_TScaleV + LeftDVZ *
subY;
    LeftRZ = RR_VA[Left-1]->RZ + LeftDRZ * subY;
}

void RR_GetRightSection()
{
    long cy = RR_Y[Right],ny;
    do
    {
        if (!Right) Right = RR_NumVerts-1;
        else --Right;
        ny = RR_Y[Right];
    } while (ny==cy);

    HRight = ny-cy;

    RightDX = (RR_VA[Right]->PX - RR_VA[Right+1]->PX) /
(RR_VA[Right]->PY - RR_VA[Right+1]->PY);
    RightX = RR_VA[Right+1]->PX + (cy-RR_VA[Right+1]->PY)
* RightDX;
}

void Referance_Rasterizer_Polygon()
{
    // Calculate Polygon Height
    dword y = ceil(RR_VA[0]->PY);
    dword my = y,My = y,mi = 0,Mi = 0,i;
    RR_Y[0] = y;
    for(i=1;i<RR_NumVerts;++i)
    {
        y = RR_Y[i] = ceil(RR_VA[i]->PY);
        if (y<my) {my = y; mi = i;}
        else if (y>My) {My = y; Mi = i;}
    }
    if (My==my) return; // zero height

    RR_Y[RR_NumVerts] = RR_Y[0];
    RR_Y[-1] = RR_Y[RR_NumVerts-1];

    Left = Right = mi;
    HLeft = HRight = 0;
    Scanline = (word *)g_VS.FB + my * (g_VS.Pitch>>1);

    // Calculate Gradients (this has degenerate cases as
well but are not likely to
    // appear, and gradients are constant anyway)

```

```

/*if (Mi-mi==1)
    RRGrads(RR_VA[mi],RR_VA[Mi],RR_VA[Mi+1]);
else
    RRGrads(RR_VA[mi],RR_VA[mi+1],RR_VA[Mi]);*/
// Use constant Gradients (works!) [not needed using
mapper/interface]
/*drzdx_ = drzdx;
drzdx_16 = drzdx_ * 16.0f;
duzdx_ = duzdx[g_lvldiff];
duzdx_16 = duzdx_ * 16.0f;
dvzdx_ = dvzdx[g_lvldiff];
dvzdx_16 = dvzdx_ * 16.0f;*/

// Initialize Outer loop function
RR_OuterInit();
do
{
    if (!HLeft) RR_GetLeftSection();
    if (!HLeft) break;
    if (!HRight) RR_GetRightSection();
    if (HLeft < HRight)
    {
        RR_SectionHeight = HLeft;
        HRight -= HLeft;
        HLeft = 0;
    } else {
        RR_SectionHeight = HRight;
        HLeft -= HRight;
        HRight = 0;
    }
    // Execute outer
    RR_OuterLoop();

    if (HLeft)
    {
        LeftX += LeftDX * RR_SectionHeight;
        LeftUZ += LeftDUZ * RR_SectionHeight;
        LeftVZ += LeftDVZ * RR_SectionHeight;
        LeftRZ += LeftDRZ * RR_SectionHeight;
    }
    if (HRight)
    {
        RightX += RightDX * RR_SectionHeight;
    }
} while (1);
}

```

```

void Reference_Rasterizer(Vertex **VA)
{
    if (RR_TexturePtr == NULL) return;

    dword i,j;

    if (1)//GetAsyncKeyState('R'))
    {
        RR_NumVerts = C_NumVtx;
        memcpy(RR_VA,VA,C_NumVtx*sizeof(Vertex *));
        RR_VA[C_NumVtx] = RR_VA[0];
        RR_VA[-1] = RR_VA[C_NumVtx-1];
        Reference_Rasterizer_Polygon();
    } else {
        RR_VA[0] = VA[0];
        j = C_NumVtx;
        C_NumVtx = 3;
        for(i=2;i<j;++i)
        {
            RR_VA[1] = VA[i-1];
            RR_VA[2] = VA[i];
            //Texture_Mapper(F,RR_VA);
            //Rasterizer(F,RR_VA);
        }
        C_NumVtx = j;
    }
}

```

```

// this enables function selection using arrays. faster,
less space consuming, but harder to read.
//typedef void (*RR_OuterInitType)();
//typedef void (*RR_OuterLoopType)();
//RR_OuterInitType RR_OuterInitFunctions[] =
{RR_OuterInit_Standard, RR_OuterInit_Interface};
//RR_OuterLoopType RR_OuterLoopFunctions[] =
{RR_OuterLoop_Standard, RR_OuterLoop_Interface};

```

```

void RR_SetTextureMapper(eTextureMappers TM)
{
    switch (TM)
    {
        case TMapper_Standard:
            RR_OuterInit = RR_OuterInit_Standard;
            RR_OuterLoop = RR_OuterLoop_Standard;
            break;
    }
}

```

```

        case TMapper_Interface:
            RR_OuterInit = RR_OuterInit_Interface;
            RR_OuterLoop = RR_OuterLoop_Interface;
            break;

    }

}

void RR_SetTextureInner(eTextureInners TI)
{
    switch (TI)
    {
        case TInner_Standard:
            RR_InnerLoop = RR_InnerLoop_Standard;
            break;
        case TInner_ZeroColorKey:
            RR_InnerLoop = RR_InnerLoop_ZCK;
            break;
    }
}

```